The University of Alberta

Abstracting and Using Domain Relations to Expedite Concept Learning

by

Peter Kin Wing Leung ©

A Thesis
Submitted to The Faculty of Graduate Studies and Research
in Partial Fulfillment of the Requirements for the Degree
of Master of Science

Department of Computing Science

Edmonton, Alberta
Fall, 1989

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Peter Kin Wing Leung

TITLE OF THESIS: Abstracting and Using Domain Relations to Expedite Concept Learning

DEGREE: Master of Science

YEAR THIS DEGREE GRANTED: 1989

Permission is hereby granted to The University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis or extensive extracts from it may be printed or otherwise reproduced without the author's permission.

Peter Leung

36 Clancy Drive
WILLOWDALE, ON
M2J 2V8

Date: Aug 28, 1989

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled Abstracting and Using Domain Relations to Expedite Concept Learning submitted by Peter Kin Wing Leung in partial fulfillment of the requirements for the degree of Master of Science.

Renée S. Elio
(Supervisor)

R. Goebel

Emil Girczyc

Francis Jeffry Pelletier

Date: Aug 17, 1989

To

Students in Tian An Men Square

on June 4, 1989

Abstract

Learning by example can be viewed as a search process. To guide this search, the learner can use a domain model that consists of constraining relations among the values of domain attributes. This research focuses on how a learner can abstract and subsequently use these domain relations to advance the simultaneous learning of multiple concept definitions in a domain. This paradigm is explored through a system called Ledora that uses the version space method as its basic concept learning algorithm. Ledora's behavior can be characterized by a "observe-theorize-experiment" framework. In observational mode, Ledora receives pre-classified concept instances provided by the environment, and maintains certain co-occurrence patterns among attribute values. In theory mode, Ledora actively verifies and generalizes domain relations suggested by these patterns. In experimental mode, Ledora uses verified domain relations to generate its own test instances, and asks the environment to classify these instances. Experiments indicated that Ledora managed a greater degree of concept learning in a fixed number of trials with this method than if it had focused *all* its attention on only observing instances presented by the environment. In addition, abstracting and using domain relations to generate test instances yields a reasonable rate of learning when compared with a method based only on the structure of attribute values and the current concept descriptions defined in terms of this structure, with the added benefit of having learned the domain relations along the way. Ledora shows a paradigm for integrating two types of learning: learning by passive observation of examples, and learning by active discovery and experimentation. We discuss the merits, assumptions, and applicability of this paradigm to particular learning scenarios and real world domains.

Acknowledgements

I wish to express my gratitude to my thesis supervisor Dr. R. Elio. Dr. Elio's ideas, and guidance were crucial to this research. I would also like to acknowledge the financial support from Dr. Elio's NSERC Operating Grant A0889.

I'd like to thank the members in my thesis examination committee: Dr. R. Goebel, Dr. J. Pelletier, and Dr. E. Girczyc. Their suggestions had greatly improved the clarity of this thesis.

I also thank my peers in the Department of Computing Science for their help and advice, especially Raymond Cheng, Brant Coghlan, Daniel Lanovaz, Yan Li, Peternela Scharf, and Bonita Wong.

I am grateful to Dr. David and Garland Yip, Vicki Lam, Doris Chai, Alex Lee, and Pauline Lim for their support and encouragement.

Most of all, I would like to thank my parents and my sister, Mrs. Alice Pittman, for their unconditional love and patience.

Table of Contents

Chapter Page

1. Introduction 1

2. Literature Review 8

    2.1 Learning as Search 8

    2.2 Discovery Systems 12

        2.2.1 BACON 12

        2.2.2 GLAUBER/NGLAUBER 14

        2.2.3 STAHL 18

        2.2.4 Brown's Theory Formation System 20

        2.2.5 Meta-DENDRAL 21

        2.2.6 AM 24

        2.2.7 Domain Knowledge in Learning 25

        2.2.8 Summary 28

    2.3 Efficiently Representing the Search Space: Version Space Method 29

        2.3.1 Candidate Elimination and Version Spaces 31

        2.3.2 LEX 36

        2.3.3 Midpoint Method 38

        2.3.4 Summary of the Version Space Method 39

    2.4 Discussion of Abstracting and Using Domain Knowledge 39

3. Predictive Relations in a Structured Domain 41

    3.1 Structured Knowledge about Domain Attributes 41

    3.2 Semantics of Predictive Relations 42

    3.3 Detecting Predictive Relations 43

    3.4 Summary 47

4. Learning Framework Design 49
  4.1 Overview 49
  4.2 World Simulator: Defining the Domain 50
  4.3 Representation of Concepts and Instances 51
  4.4 Maintaining Co-occurrence Information 52
  4.5 Monitoring Consistency of Predictive Relations 55
  4.6 The Observe-Theorize-Experiment Framework 57
    4.6.1 The Applicability of Predictive Relations 60
      4.6.1.1 Applicability Constraints on the Predictable 61
      4.6.1.2 Applicability Constraints on the Predictor 61
    4.6.2 Observational Mode 66
    4.6.3 Theory Mode 67
      4.6.3.1 Testing Prediction Credibility 67
      4.6.3.2 Generalizing Predictive Relations 71
    4.6.4 Experimental Mode 74
      4.6.4.1 Running the Experiment 75
      4.6.4.2 Guaranteeing Improvement over Midpoint Method 78
  4.7 Summary 79
5. Implementation Details 80
6. Experiment Results 85
  6.1 Overview 85
  6.2 Dependent Variables 87
  6.3 Pilot Experiments 89
    6.3.1 Pilot-1 89
    6.3.2 Pilot-2 92

6.4 Effects of Presentation Order 96
6.5 Chaining Effects and Interactions with Presentation Order 107
6.6 Summary 110
6.7 Control Experiments 110
6.8 Simplifying System Parameters 116
7. General Discussion 120
7.1 Generality of the Observe-Theorize-Experiment Framework 121
7.2 Level of Instance Description 123
7.3 Evidence for Predictive Relations 124
7.4 Truthfulness of Predictive Relations 126
7.5 Usefulness of Predictive Relations 127
7.6 Importance of Testing Predictive Relations 129
7.7 Relations to Other Work 130
7.8 Future Work and Extensions 133
7.8.1 Further Experimentation 133
7.8.2 Modifying Generalization Hierarchies 133
7.8.3 Design Changes 135
7.9 Summary and Conclusions 135
Bibliography 137
Appendix: Generalization Hierarchies 141

List of Figures

Figure Page

2.1 A Rule Discovered by Meta-DENDRAL.......23

2.2 Generalization Hierarchy for Occupation-of-Victim.......31

2.3 Candidate Elimination Algorithm.......32

2.4 Generalization Hierarchies of Two Attributes.......34

2.5 Some Rewrite Rules in LEX.......36

3.1 Generalization Hierarchy for the Weapon Attribute.......42

3.2 Relevant values and frequency counts for testing.......45

4.1 Interactions between World Simulator and Ledora.......51

4.2 Snapshots of the Frequency Table.......54

4.3 Ledora's Control Structure.......59

4.4 Partial Generalization Hierarchy for Crime-Site.......63

4.5 Partial Generalization Hierarchy for Type-of-Activity.......77

6.1 Concepts and Predictive Relations used in Pilot Experiments.......90

6.2 Results of Experiments Pilot-1 and Pilot-2.......91

6.3 Revised Target Concept Definitions.......95

6.4 Predictive Relations in Chaining & No-Chaining Experiments.......97

6.5 Results for Iterative versus Random Concept Presentation.......98

6.6 Timetable for the Discovery of Relations in I-C and R-C/R-C'.......102

6.7 Mode Activities in Experiment I-NC.......106

6.8 Mode Activities and Concept Order in Experiment R-NC.......106

6.9 Mode Activities in Experiment I-C.......108

6.10 Control Experiments versus Others.......113

6.11 Simplified versus Non-Simplified Parameters.......119

7.1 Bias Before and After Shift.......134

Chapter 1

Introduction

The ability to learn is a fundamental characteristic of both human and machine intelligence. Since the very beginning, Artificial Intelligence (A.I.) researchers have proposed and studied many diverse methods for providing computers with this ability. Learning by example, sometimes known as concept learning, is one of the earliest and most studied methods for machine learning. Given a set of positive examples and negative examples of some concept, the learner formulates a concept description that describes all the positive examples but none of the negative examples [Carbonell, et al., 1983].

The problem of learning by example can be viewed as a search problem [Mitchell, 1982]. Specifically, states in the search space correspond to the alternative descriptions of a domain concept. Given a positive or negative example of a concept, inductive operators such as generalization and specialization generate new states. The goal state corresponds to the final concept description.

Often, the search space for learning a concept is very large. In these cases, efficiency of search becomes a serious concern. Search should be guided along paths that lead to the goal state and away from paths that do not lead to the goal state. This raises a very important question: "What kind of knowledge is available to the learner to guide this search?" One such kind of knowledge is structural knowledge of values that domain attributes may take. For example, suppose that the attribute "shape" can take on any of the following values: "any-closed-figure", "polygon", "square", and "triangle". Structural knowledge in this case includes polygon is more general than square and triangle. This kind of knowledge allows the learner to efficiently and simply represent what it has learned about specific concepts to date. For instance, the shape attribute in the concept description might be only partially-known as being more specific than "any-closed-figure" but more general than "square". Another kind of knowledge is a *model* of the domain. This domain model may consist of rules about the interactions and dependencies among the values of domain attributes. All concept descriptions and all instance descriptions must obey, either frequently or always, the constraining rules in this model. One type of rule is a simple *co-occurrence relation*, for instance, the value "small" of attribute "size" and the value "light" of attribute "weight" often appear together in the same description across many different objects. Another type of potentially more powerful rule describing attributes is *predictive relations*, where the value of one attribute predicts the value of a second attribute. For instance, one such relation in the domain of "consumerism" may be this: If the availability attribute of an item has value low, then that item's cost attribute has value high.

A system designer often does not explicitly give a domain model to the learner. However, by observing instances of several different concepts in a domain, the learner may infer relations among attribute values that seem to hold true in the domain, across all concept descriptions. For instance, the predictive relation "low availability predicts high cost" might stem from learning descriptions of specific consumer goods (mink coats, jewelry, Pacific salmon, etc) in which availability and cost are part of the concept descriptions.

A domain model can guide the search process by effectively pruning portions of the search space. This leads to faster learning of domain concept descriptions. For example, in learning the description of the concept "rare-coins," the learner can exploit its knowledge that low availability predicts high cost in the following manner. Knowing that rare-coins in general have low availability, all possible descriptions of rare-coins in the search space in which cost is *not* high can be eliminated.

This research focuses on how a learner can abstract and subsequently use predictive relations to advance the simultaneous learning of multiple concepts in a domain. We are essentially exploring a dual learning task: acquiring both a set of target concept descriptions and a set of predictive relations that characterize the domain and can be used to learn the target concept descriptions. While the primary task is still to learn the target concept descriptions, the learner abstracts information about predictive relations from instances of the different concepts it is studying. Some predictive relations may be relevant to the target concepts that the system must learn. Thus, if low availability seems to predict high cost, and the attributes availability and cost are involved in several as-yet-unlearned concepts, it would be useful to verify that this predictive relation does generally hold true in the domain. In this case, the learner suspends its primary task of learning concept descriptions and turns its attention to verifying and refining this predictive relation.

Since abstracting and verifying predictive relations use up valuable resources (as measured by learning trials) that could otherwise be spent on learning the domain concepts, one might question whether predictive relations are worth the effort. In our attempt to answer this question, we compare the degree of learning of the entire set of domain concepts after a fixed number of trials in both experiments with and without learning predictive relations. Our thesis is that the additional effort for acquiring and verifying predictive relations is ultimately worth it because, with these predictive relations, the concepts are overall more completely learned after a fixed number of trials.

The specific concept learning algorithm that we use for our investigation is the version space method [Mitchell, 1982]. This method is based on a data structure called a version space that efficiently represents all the plausible "versions" of the concept description that are consistent with the instances seen so far. The representation of a version space requires that the values of the domain attributes be organized in generalization hierarchies. Essentially, these hierarchies capture structural knowledge of the attributes' values. Using these hierarchies, a version space can be represented by two boundary sets: the maximally-general boundary set (*G*) and the maximally-specific boundary set (*S*). The algorithm used in conjunction with this data structure is the candidate elimination algorithm. In this algorithm, positive and negative instances of the concept update these boundaries so that only a single version remains in this space, and this version corresponds to the final concept description.

The version space approach to concept learning allows the learner to use what we call the "midpoint method" to generate its own test instances of domain concepts, and receive feedback from the environment on whether they are positive or negative. To generate a test instance, the midpoint method uses knowledge of the generalization hierarchies and of the current boundaries of the version spaces. Specifically, the learner constructs a test instance by instantiating a randomly-chosen attribute with a value that splits the version space into two halves. The midpoint method guarantees that each instance generated will reduce by half the version space for the chosen attribute, regardless of whether the instance is positive or negative.

The control structure for learning multiple concepts using the midpoint method is very simple. Suppose that there are N concepts to be learned with M attributes each. Beginning with attribute j of concept i, the learner generates test instances, using the method outlined above, until that attribute's final value is determined. Then, it continues generating test instances to determine, one by one, the final values for the rest of the attributes of concept i, that is, attributes j+1, j+2, etc. Next, it repeats the above procedure for concepts i+1, i+2, etc, until all concepts have been learned. This method keeps generating test instances with value choices that systematically converge one attribute at a time for each concept to be learned.

Our goal is to investigate whether the instances generated using predictive relations are more informative than those generated using the midpoint method. Being more informative means that, regardless of whether the instance is classified as positive or negative, it provides more information about the final concept description. In terms of search, this means a larger section of the version space is pruned as a result of the instances generated using predictive relations. This raises various control issues. Given that, at any time, the learner can use different predictive relations to generate instances of different concepts, and, more importantly, that a different amount of pruning may result due to the predictive relation and concept chosen, the learner must now decide which predictive relations seem most useful, how they can be verified, and for which concepts they should generate test instances.

To examine the above issues, we designed and tested a system called Ledora (Learning by experimenting on domain relations). Ledora's task is to learn multiple concepts simultaneously in a specific domain. It does not know in advance how many concepts it must learn; it becomes aware of a new concept when it encounters an instance of a concept that it has never seen before. The world with which Ledora interacts is defined by a set of structured attributes that characterize the concepts in the domain and their instances, a set of target concept descriptions, and a set of target predictive relations that constrain the values that attributes may take in this world.

There are three ways that Ledora could learn domain concept descriptions. First, the environment supplies Ledora with examples that are pre-classified (i.e., positive or negative), and Ledora simply updates the version spaces with these examples. Second, Ledora generates test instances using the midpoint method, and asks the environment for their classification. Third, Ledora uses domain-specific predictive relations it has abstracted from observing instances of several different concepts to generate test instances, and asks the environment to classify them. These instances, generated with a domain model, are more informative than the midpoint method.

Ledora's behavior in terms of abstracting, verifying and using predictive relations best resembles a scientist whose task is to discover new knowledge in a certain field. Its activities are organized into a "observe-theorize-experiment" framework. In observational mode, the system observes pre-classified concept instances provided by the environment, modifies the version space of the appropriate concepts, and maintains information about potential predictive relations. In theory mode, Ledora actively verifies and generalizes (if possible) predictive relations that it has recognized while observing instances. In experimental mode, Ledora uses predictive relations that have been verified in theory mode to generate its own test instances for domain concepts. It then asks the environment to classify these test instances.

Our preliminary experiments with Ledora indicate that although Ledora uses up resources (measured as instances asked about or received) in theory mode, it still managed a greater degree of domain concept learning in a fixed number of trials than if it had focused *all* its attention on observing instances presented by the environment. In addition, abstracting and using predictive relations to generate test instances yields a reasonable rate of learning when contrasted with the midpoint method, with the added benefit of having learned the predictive relations along the way.

The role and significance of predictive relations in this study is twofold. First, using predictive relations to generate test instances is an improvement over the midpoint method under Mitchell's version space paradigm. This improvement is measured in terms of learning speed. Specifically, we measure the degree of learning in a fixed number of training instances (either observed or self-generated). However, the more important aspect of this work is that Ledora shows a new approach of integrating two types of learning: learning by passive observation of examples, and learning by active discovery and experimentation. On one hand, the examples used in learning the domain concepts constitute the input data of a discovery process aimed at discovering higher-level domain relations. On the other hand, the discovered domain relations are subsequently used to generate test examples for learning the domain concepts. This approach of integrating discovery with learning by examples can be regarded as a broader paradigm for discovery systems.

Chapter 2

Literature Review

This chapter is organized into three sections. The first section reviews the general framework of learning as a search process, and introduces important terms and issues in this framework that we use in the remaining two sections. Because detecting predictive relations is essentially a discovery learning process, we review, in the second section, the major discovery systems. The third section focuses on learning by example as a search process. Specifically, we describe a particular method, Mitchell's [1977] version space and candidate elimination algorithm, designed to represent and update efficiently the space of concept hypotheses as new examples are seen. Ledora uses this method in conjunction with the predictive relations it discovers to learn the definition of domain concepts.

2.1 Learning as Search

The problem of inductive learning can be viewed as a search problem. Specifically, learning by example (one form of inductive learning) can be mapped to the search framework. States in the search space correspond to alternative concept descriptions. The goal state is the concept description that covers all positive examples of the concept and rejects all negative examples. Operators correspond to inductive processes such as generalization and specialization. The search space framework is useful for understanding two important distinctions in learning: incremental versus non-incremental learning, and data-driven versus model-driven learning.

In the context of learning by example, incremental learning means that the learning system receives concept examples one at a time. For each example, the learning system adjusts its current concept description, if necessary, through generalization or specialization processes so that it is consistent with the example. This process corresponds to moving to another state in the space by applying an operator. It then discards the example, disallowing any future reexamination of this example. Hence, if more than one alternative concept descriptions are consistent with a given example (more than one way to generalize or specialize), the learning system can be designed to keep a set of alternative descriptions, i.e., maintain a set of states in the search space for later expansion. The most notable example of this type of system is Winston's [1975] program for learning structural descriptions. Other researchers have also explored this paradigm (e.g., Mitchell, et al. [1983], and Jones [1986]).

In contrast, a non-incremental learning system has all the training examples at its disposal throughout the learning experience. Hence, it may re-process some or all the concept examples at any time. INDUCE 1.2 [Dietterich & Michalski, 1981] is a good example of a non-incremental learning algorithm. After arriving at a state (i.e., alternative concept description) in the search space, the entire set of training examples are used to evaluate this state. Specifically, the learner computes the number of training examples that the concept description matches. If the resultant number does not reach a minimum requirement, then this state is pruned, and the learner backtracks to a previous state in the search space.

The search framework clarifies the distinction between data-driven and model-driven learning.[1] Data-driven learning means that the training examples (or data) alone determine the hypotheses (i.e., alternative concept descriptions) that the learning system constructs. In the search context, the data drive the learning system from one state to another, in that an inductive operator is chosen to reconcile the current concept description with the data, thus moving to a new state.

In model-driven learning, domain knowledge (a term we use interchangeably with domain model) influences what concept descriptions are considered by the learning system. A domain model effectively directs the search to some area of the space and away from other areas. The main role of examples in model-driven learning is to confirm (or deny) alternative concept descriptions; in data-driven learning, examples functionally generate the new concept description. In model-driven learning, only *some* training examples may be examined by the learning system in order to suggest a concept description under the constraints set by the domain model. Then, the entire set of training examples is used to confirm or deny that proposed concept description or refinement to the concept description. In data-driven learning, the examples are the only source of the hypotheses. In other words, the decision of which area to search in the space is solely determined by the examples.

---

[1] The data-driven versus model-driven distinction is also relevant in problem-solving.

Generally speaking, then, domain knowledge influences the exploration of the search space. Domain knowledge may not serve as a model in the strict sense, i.e., it is not the source of the hypotheses. However, it can serve the same role as an evaluation function, i.e., it can answer the question, "Given my domain knowledge, is this state (possible concept description) in the search space a reasonable one?"

One advantage of model-driven learning is that it is less sensitive to noise. The reason for this property is twofold. First, not all the data are used to generate the hypotheses. Thus, minor discrepancies among examples due to noise do not bounce the system from one concept description to another. Second, a hypothesis is often accepted (rejected) if only *most* of the data confirm (deny) it. Therefore, variations in a particular example due to noise may not cause the concept description to be abandoned. In comparison, data-driven learning is quite sensitive to noise since each example has its impact on what the final concept description looks like. A single example, if corrupted via noise, can prune a subsection of the search space that may actually contain a good or correct concept description.

The advantage of model-driven learning can also be viewed as a disadvantage: because it defines or constrains what areas of the search space to consider, domain knowledge represents a kind of "bias" in what can be ultimately learned. A bias, according to Utgoff [1986], is any factor, other than training examples of the concepts being learned, that influences hypotheses selection. The effect of domain knowledge on model-driven learning is this: although there might be better alternative concept descriptions, the learning system can only learn those descriptions that adhere to the constraints set by the domain knowledge.

A bias can influence hypotheses selection by affecting how the search space is defined or how it is searched. An example of a bias that affects the definition of the search space is the representation language in which concepts (and instances) are described. Because a learning system can only learn what it can describe, the representation language used for the domain constitutes a form of bias. This bias directly influences the definition of a search space by determining what can and what cannot be included in the space. Bias in the search procedure itself includes any heuristics used to drive the exploration of the search space. The biases together with the training examples completely determine how alternative concept descriptions are derived from the training set of examples.

In both learning-from-example systems and learning-by-discovery systems, we can locate some kind of bias. In addition, we can identify whether that bias is, more or less, domain-independent or domain-specific. In the following section, we review several important discovery systems in the context of the search framework. As we shall see, discovery systems that function without direct environmental supervision include bias to help them focus on sound and/or meaningful hypotheses.

2.2 Discovery Systems

Learning by observation and discovery is one form of learning that searches, without the direct supervision of a teacher, for regularities or a theory that explains the observed data. In such learning, the discovery system is not provided with instances pre-labelled as positive or negative for specific concepts. To obtain the data, most discovery systems interact with the external environment to some degree [Carbonell, et al., 1983]. The degree of interaction can vary from passive observation, where the discovery system relies solely on the environment to provide the inputs, to active experimentation, where the discovery system simulates experiments by perturbing the environment in order to verify its hypotheses. In the following, we first review the seminal discovery systems. Then, we discuss the end-product of discovery learning: domain knowledge. This is important because Ledora engages in a kind of discovery by experimentation in order to learn domain concepts faster.

2.2.1 BACON

BACON [Langley, et al., 1986] is a system that discovers quantitative chemical laws. The input to BACON describes a variable in terms of its name, type (independent or dependent), and allowable values. Essentially, BACON can be viewed as running experiments: BACON specifies the values of the independent variables in each experiment, and asks the user/teacher to provide the values of the dependent variables. BACON uses a factorial design of experiments, i.e., the experiments include all possible combinations of values of the independent variables. Using the values of both independent and dependent variables, BACON generates one or more functions relating these variables. For example, suppose that the pressure P, the temperature T and the quantity N of a gas are three independent variables and that the volume V is a dependent variable. After a series of experiments, BACON outputs PV = 8.32 N (T + 273), which is also known as the ideal gas law.

BACON's behavior is best described as successively generating higher levels of data description. A level of data description includes the values of some independent variables and some dependent variables. At each higher level of data description, a function is formulated that relates *one* remaining independent variable with the dependent variables. The generation of higher-level description terminates when a function is found that relates all independent variables with the dependent variables.

Using the ideal gas law as an example, BACON first varies one of the independent variables (say, P) while holding the rest of the independent variables (N and T) constant. BACON then searches a set of functions between the chosen independent variable (P) and the dependent variable (V) using a method that is described below. For example, BACON hypothesizes that $V^{-1} = 0.000425$ P when T is 30 and N is 1. BACON repeats the above experiments, i.e., varying the values of P, but this time choosing a different constant for a second independent variable, say T. To illustrate, BACON notices that $V^{-1} = 0.000410$ P when T is 20 and N is 1. After repeating the experiments with all possible values of T, BACON has a set of values for the coefficient parameter (*a*) of the function, $V^{-1} = aP$. This set of values includes 0.000425 and 0.000410.

The coefficient parameters of the function derived at one level of description are regarded as dependent variables at the next higher level of description. Essentially, the values of these parameters, which are BACON's hypotheses summarizing a lower-level data description, are used as data to generate the next higher level of description. For instance, 0.000425 and 0.000410, coefficients of the function found in the first level of description, become values of a dependent variable at the second level of description.

The process of generating successively-higher levels of description continues until all the independent variables are incorporated into the function.

BACON uses a model-driven approach in its search of a function at a given level of description. The search is constrained by a set of templates (or forms) of arithmetic functions among variables, for instance $Y^{-1} = aX$. Although Langley et al. [1986] did not characterize them as such, one can view these templates as representing typical ways that variables interact in the physical world. As such, they are quite domain-specific. Considering that there are an infinite number of possible functions relating some set of variables, these templates make BACON's task quite well-defined, namely, find and instantiate the function template that best describes the observed data. In this light, the templates constitute a bias by defining completely the way in which BACON attempts to summarize the observed data.

Thus, states in BACON's search space are these function templates instantiated with particular values. The goal state is a function that maximally predicts the observed data. Operators correspond to incrementing and decrementing the values of the parameters in a function. In comparing two alternative sets of parameter values (two possible paths out of a particular state), BACON prefers the one that best predicts the observed values of the variables involved in the function. In other words, the observed data is used as an evaluation function to determine the goodness of a given state. In this way, observed data is only used to test hypotheses (alternative ways to instantiate a fixed number of functions) and not to generate them. Hence, BACON can handle some noise in the data.

2.2.2 GLAUBER/NGLAUBER

Unlike BACON, both GLAUBER [Langley, et al., 1986] and its successor NGLAUBER [Jones, 1986] discover qualitative, rather than quantitative, laws in the chemistry domain. GLAUBER's input consists of a set of qualitative facts. Each input fact contains a relation name and a set of attribute-value pairs. For example, the fact that HCl reacts with NaOH to form NaCl is represented as

(reacts inputs {HCl NaOH} outputs {NaCl}).

Given these facts, GLAUBER's task is to define extensionally classes of chemical substances mentioned in the facts, and to formulate qualitative laws having the same format as the input facts but with specific substances replaced by the more general classes. For instance, GLAUBER discovers the law (reacts inputs {acids alkalis} outputs {salts}), where acids, alkalis and salts are classes.

GLAUBER is a data-driven, non-incremental discovery system. The initial state in GLAUBER's search space is the set of input facts containing only specific chemical substances (or constants). The goal state consists of generalized statements relating some classes, along with the class definitions. GLAUBER has two inductive operators: FORM-LAW and DETERMINE-QUANTIFIER. At a high level, these two operators can be viewed as replacing constants with variables. Given a set of facts that have the same relation and have at least one common substance, the operator FORM-LAW replaces these facts with a single law in which some specific substances in the facts are replaced with a new class name. Any new class is defined to contain the corresponding substances that it replaced in the facts. For example, GLAUBER summarizes the two facts

(reacts inputs {HCl NaOH} outputs {NaCl}), and
(reacts inputs {HCl KOH} outputs {KCl})

by introducing two classes, say *x* and *y*, for {NaOH KOH} and {NaCl KCl} respectively, and conjecturing the law (reacts inputs {HCl x} outputs {y}). Hence, these class names are like variables, in that any member of the class satisfies the law.

The operator DETERMINE-QUANTIFIER determines whether each class mentioned in the law should be existentially or universally quantified. GLAUBER determines the quantifiers by checking the given facts. Continuing from the above example, GLAUBER applies DETERMINE-QUANTIFIER to (reacts inputs {HCl x} outputs {y}) to obtain the quantified law ∀x ?y (reacts inputs {HCl x} outputs {y}). The first quantifier in the law, ∀, follows from the class definition; the second quantifier, ?, denotes that the precise quantifier (∀ or ∃) must be determined empirically. Specifically, *y* is universally quantified if the given input facts include all of the following:

(reacts inputs {HCl NaOH} outputs {NaCl}),
(reacts inputs {HCl NaOH} outputs {KCl}),
(reacts inputs {HCl KOH} outputs {NaCl}), and
(reacts inputs {HCl KOH} outputs {KCl}).

Otherwise, *y* is existentially quantified.

For any set of input facts, GLAUBER can apply its operators to define many different classes and hence formulate many alternative laws. GLAUBER's search is biased toward selecting the substance - HCl in the above example - that is common to the largest number of facts with the same relation. Essentially, this bias amounts to an evaluation function that prefers generating larger classes over smaller classes.

NGLAUBER [Jones, 1986] differs from GLAUBER, its predecessor, in two ways. First, it accepts input facts incrementally. Second, because input is incremental, the hypotheses that NGLAUBER makes may be refuted by subsequent input facts. These hypotheses are known as predictions in NGLAUBER.

A prediction is represented as a pair of statements denoted by *Prediction* and *For*, where Prediction is a specific statement to be verified, and For is a statement that is true if the Prediction is true. For example, the following is a prediction:

Prediction: (hasquality object {KCl} taste {salty}).
For: ∀salts (hasquality object {salts} taste {salty}).

By making this prediction, NGLAUBER's claim is this: it knows that all salts taste salty if it knows that KCl tastes salty, because KCl is defined to be a salt.

Formulating a prediction can be viewed as an operator that immediately replaces constants with variables on the basis of a single observation. For instance, suppose that x is a known class containing block1 and block2, and that the next fact encountered indicates that block1 is a cube. NGLAUBER makes the following prediction:

Prediction: (hasquality object {block2} shape {cube})
For: ∀ x (hasquality object {x} shape {cube}).

Subsequent input facts are constantly monitored to see if they confirm or disconfirm the Prediction part of any prediction. For example, if NGLAUBER later encounters a fact stating that block2 is indeed a cube, then it asserts the For part of the prediction, i.e., ∀x (hasquality object {x} shape {cube}), as a fact in its memory. Instead, if the fact indicates that block2 is a cylinder, then NGLAUBER removes the prediction altogether. NGLAUBER drops a prediction based on the observation of a single fact because, considering its domain of chemistry, physical laws either hold or they don't. In domains where generalizations may hold true most, but not all, of the time, or where generalizations may result from biased data and hence may turn out to be false later, a weakening mechanism could be used. Instead of removing the prediction, the learning system would weaken the associated strength of the prediction.

By making predictions, NGLAUBER is effectively running experiments. Specifically, the purpose of these experiments is to find out from the environment whether or not generalized statements (i.e., the For parts in predictions) created by replacing constants with classes hold true. The ability to make predictions allows NGLAUBER to be an active experimenter rather than a passive observer.

2.2.3 STAHL

STAHL [Zytkow & Simon, 1986] is a discovery system that determines componential models from qualitative information about chemical reactions. The input consists of an ordered list of chemical reactions, represented in the same format as in GLAUBER, for instance, (reacts inputs {hydrogen oxygen} outputs {water}). The output is a set of componential models of compounds that appear in the reactions. For instance, STAHL discovers the model of water, represented as (components of {water} are {hydrogen oxygen}).

Each state in STAHL's search space corresponds to a set of componential models that STAHL has discovered so far. The initial state is the null set. The goal state contains one componential model for each compound mentioned in a reaction. To move from one state to another state, STAHL uses one or more of its operators: INFER-COMPOSITION, REDUCE or SUBSTITUTE. The most basic operator is INFER-COMPOSITION, which can be stated as follows: If *A* and *B* react to form *C*, or if *C* decomposes into *A* and *B*, then infer that *C* is composed of *A* and *B*. The REDUCE operator can be expressed as this: if *A* occurs on both sides of a reaction, then remove *A* from the reaction. The third operator is SUBSTITUTE: If *A* occurs in a reaction, and *A* is composed of *B* and *C*, then replace *A* in this reaction with *B* and *C*. These operators are very algorithmic and domain-specific ways of manipulating chemical reactions.

STAHL is a data-driven learning system that processes input reactions one at a time. Given a chemical reaction, STAHL performs essentially a best-first search. First, STAHL applies its operators to this reaction in all possible ways, creating one or more states in the search space. Each new state contains the componential models in the parent state and possibly one *additional* componential model derived using the current reaction. If a new state does not contain an additional model derived from the current reaction, STAHL removes this state from the search space. Moreover, if none of the new states contain such an additional componential model, then STAHL removes all the new states, and saves the current reaction in a list for re-processing later when more componential models are available. For each new state that *does* contain an additional componential model, STAHL checks the model for consistency with other models in the state. For example, STAHL recognizes an inconsistency error if it finds both (components of {A} are {B C}) and (components of {A} are {B C D}). We can view this use of existing componential models as an evaluation function of how good a given state is. The accumulating componential models constitute a bias as to which part of the search space that STAHL will explore. This bias becomes stronger as the number of known componential models increases. In other words, STAHL is performing a more informed search over time.

If more than one state contains a consistent componential model derived from the current reaction, STAHL selects the state with the best such model, defined as one that subsumes the corresponding models in the other states. STAHL uses a very algorithmic way to compare two models, say, M_1 and M_2: substitute M_1 into the current reaction, and then repeat the same operator sequence that previously resulted in M_2. If no inconsistency is detected, then M_1 subsumes M_2.

An inconsistency between a new model and an existing model can happen if the chemical reactions presented to STAHL are noisy. If none of the new states contain a componential model inferred from the current reaction that is consistent with existing models, STAHL removes these states from the search space. It then retrieves all original reactions involving substances in the inconsistency, and re-applies its operators to each such reaction, this time bringing to bear all the componential models it has learned to date. Because more componential models have been learned since those reactions were first considered, applying operators to the reactions usually results in consistent componential models. Hence, STAHL can handle noisy data to a certain degree by revising its theories using accumulated knowledge.

2.2.4 Brown's Theory Formation System

Brown [1973] designed and implemented a system that formulates conjectures about structural redundancies in a data set and heuristically validates those conjectures. The input to the system consists of the complete extensions of a set of binary relations. The output includes all intensional definitions for each binary relation in terms of other binary relations or itself. For example, one definition of the relation Grandparent is as follows:

Grandparent (x y) <=> Parent (x z) ∧ Parent (z y).

In short, Parent/Parent (x y).

Brown's system uses an algorithmic m[illegible]r generating the intensional definitions of a relation (e.g., Uncle). For each extension of the relation Uncle, say, (Peter John), the system finds the set of all possible "paths" from Peter to John. The set of all possible paths from Peter to John may include:

Spouse (Peter Mary) ∧ Aunt (Mary John), and
Brother-in-Law (Peter Tom) ∧ Parent (Tom John).

After finding the path set for each extension of Uncle, it computes the intersection of these sets. Only specific relations (e.g., Spouse) and not the terms (e.g., Mary) in a relation are used in computing the intersection. Suppose the path set for Uncle (Jack Jill) includes:

Husband (Jack Janet) ∧ Aunt (Janet Jill), and
Brother-in-Law (Jack Rick) ∧ Parent (Rick Jill).

Thus, the intersection of the above path sets for Uncle (Peter John) and Uncle (Jack Jill) is Brother-in-Law/Parent.

The outcome of the intersection is the conjectural definitions of the relation. Because these definitions are no more than hypotheses, their truthfulness must still be verified. One danger with these definitions is that they may be overly-general. For instance, A/B is an over-generalization for C, where A, B, C are binary relations, if there exists a triplet (x y z) such that A(x y) and B(y z) are true, yet C(x z) is not true.

Brown is concerned with verifying conjectural definitions without exhaustively checking all possibilities. The heuristic technique Brown uses involves testing the so-called "inverse image of x." The inverse image of x with respect to a binary relation R is defined to be the set $\{y \mid \exists z, R(x\ z) \text{ and } R(y\ z)\}$. A conjectural definition for Uncle, say Husband/Aunt, passes this test if for each element x of R's domain, the inverse image of x with respect to Uncle is the same as the inverse image of x with respect to Husband/Aunt. If an element is found such that the two inverse images differ, the conjectural definition is rejected. This element is a counter-example of the definition. Brown notes that control often switches back and forth between generation and verification of intensional definitions until the system arrives at some definitions that are consistent with all data.

Brown reported that his system discovered several unexpected definitions that are better than the standard definitions by some criterion. For example, one of the unexpected definitions discovered for the relation Uncle is Brother-in-Law/Parent. This definition can account for all extensions of Uncle. In terms of simplicity, it is better than the standard definition, Brother/Parent ∨ Husband/Sibling/Parent, because it involves no disjunct.

2.2.5 Meta-DENDRAL

Meta-DENDRAL [Buchanan & Mitchell, 1978; Buchanan, et al., 1976] discovers rules characterizing the behavior of a scientific instrument called a mass spectrometer. A mass spectrometer bombards a chemical compound with electrons, causing the molecules to fragment. The input to Meta-DENDRAL is composed of data points collected from the mass spectrometer. Each data point consists of (a) the known molecular structure of a given chemical compound, for instance, CH3-CH2-CH2-NH-CH2-CH3, and (b) the corresponding fragmentation activities. An example of a rule that Meta-DENDRAL may discover is:

C-C-N-C -> C*C-N-C

The left-hand side of a rule specifies the bond structure of the relevant portion of a molecule for the fragmentation activities to occur. The right-hand side illustrates the resultant fragmentation activities. The "*" symbol indicates the position of the broken bond.

Given data about how specific compounds fragment, Meta-DENDRAL's goal is to search for general fragmentation rules (or regularities) that can describe these data. Meta-DENDRAL is a model-driven system, using a so-called "half-order theory" of mass spectrometry to guide its search for rules. This theory specifies general constraints for how molecules fragment. Examples of such constraints are "Two bonds to the same carbon atom cannot break together" and "No more than three bonds break in any one fragmentation."

This half-order theory constrains the forms that a fragmentation rule may take. Each state in Meta-DENDRAL's search space is a potential rule within the constraints of the half-order theory. The initial state is the most general rule X*Y where X and Y are any atoms. This rule predicts that the bond between any two atoms will break. Meta-DENDRAL basically moves from more general versions of a fragmentation rule to more specific versions by applying a specialization operator. This operator specializes a rule within the constraints of the half-order theory, by specifying various attributes of atoms, such as their type or the number of neighboring atoms that are hydrogen. Thus, the initial rule X*Y may eventually be specialized to the form C*C-N-C with attributes attached to each atom. Figure 2.1 shows, for each atom in the final rule, the values of attributes atom type and number of hydrogen neighbors.

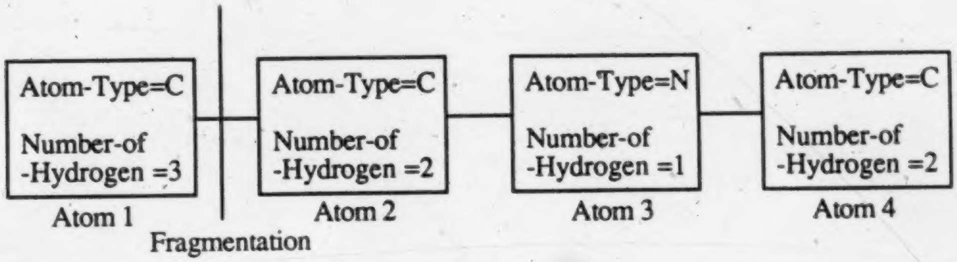

Atom-Type=C
Number-of -Hydrogen =3
Atom 1
Atom-Type=C
Number-of -Hydrogen =2
Atom 2
Atom-Type=N
Number-of -Hydrogen =1
Atom 3
Atom-Type=C
Number-of -Hydrogen =2
Atom 4
Fragmentation

Figure 2.1 A Rule Discovered by Meta-DENDRAL.

The entire set of fragmentation data is available for inspection at all times, making Meta-DENDRAL a non-incremental system. Each state generated in the rule space is checked against this data set. The "goodness" of each state is determined by the amount of data correctly predicted by the corresponding rule. Only those (more specific) states that yield a higher value than their (more general) parent state are retained. Hence, the degree of "fit" with the data set can be viewed as an evaluation function that determines whether search should continue along a particular specialization in the rule space. A goal state corresponds to a rule that has a better fit of the data than any of its specializations.

The search as described above essentially uses a Generate-and-Test strategy. The role of the half-order theory is to constrain the search of the rule space. Rule hypotheses are confirmed or disconfirmed by the data. Since the data is only used to verify the hypotheses, Meta-DENDRAL can tolerate some degree of noise in the data.

2.2.6 AM

AM [Lenat, 1979; 1982; 1983a] is a discovery system that defines and explores mathematical concepts under the guidance of a set of pre-defined heuristics. Mathematical concepts are represented as frame-like structures. An important slot in a concept's frame is the "examples" slot. AM's behavior is driven, to a large extent, by finding examples for concepts. Specifically, AM starts out filling in examples of pre-defined concepts. Having generated examples of these concepts, it is in a position to recognize regularities among these examples and propose new concepts to describe them. Once a new concept is proposed, AM sets out to fill in the empty slots of the corresponding frame, focusing first on the examples slot.

The above activities are guided by a set of powerful heuristics. Some heuristics are quite general, for example, "If something happens frequently, then explore it." Others are quite specific to the mathematics domain, for example, "If *f* is an interesting function which takes a pair of *A*'s as inputs (f:AxA), then define and study the function g(a) = f(a a)," and "If you want to find examples of some concept C with a recursive definition, then read off a trivial example from the base step of the recursion."

Characterizing the states in AM's search space is somewhat different from what we've seen for previous systems. AM is not searching for one particular concept. We can view it as searching for the most interesting enlargement of its mathematical knowledge. Thus, we can view each state in AM's search space as corresponding to a corpus of mathematical concepts. The initial state is a set of incompletely-specified concepts, i.e., concepts with some empty slots. The search space states cannot be characterized *a priori* because AM does not have a well-defined goal (other than exploring the domain) for directing its search efforts. Rather than trying to summarize data provided for it, it is effectively generating its own data and noticing patterns in the data. If AM's goal were to summarize given data, it would stop after the data is summarized. Because AM generates its own data, it can continue indefinitely.

AM's operators are the pre-defined heuristics that define new concepts or fill in slots of existing concepts. At any time, one or more operators may be applied to generate one or more states in the search space. To determine which state is to be generated next, AM uses an agenda mechanism. Each task in the agenda corresponds to a specific operator application, and is accompanied by a measure of the task's "interestingness" that is calculated in a very ad hoc way. Specifically, the "interestingness" of a task is determined by the number (and strength) of the reasons for believing the task to be worthwhile.[2] AM executes the task in the agenda that is "most interesting." Hence, the interestingness measure can be viewed as an evaluation function. It does not matter to AM which concepts are discovered or missed as long as it spends its time on interesting tasks.

2.2.7 Domain Knowledge in Learning

This section discusses one type of domain knowledge that can be used in learning, namely, relationships among attributes' values within the domain. We first review Russell's [1986] notion of "higher-level regularities," and then discuss the ways in which these regularities can be considered domain knowledge.

Russell [1986] proposes a relationship called a "higher-level regularity" as part of his theory of induction. The following example, adapted from Russell, clarifies this relationship.

> John Smith is an American tourist in Italy. On meeting the first Italian in the airport, John observes that he speaks Italian. The next several Italians John

---

[2] "Worth" and "interestingness" are terms that Lenat seems to use interchangeably. The initial set of primitive mathematical concepts have an associated worth value, and worth propagates to new concepts through an algorithm[Lenat, 1983c].

encounters also speak Italian. This prompts John to conjecture that all Italians speak Italian.

This conjecture is, in Russell's view, an example of a higher-level regularity. Higher-level regularities represent knowledge about the world abstracted from specific observations seen to date, so clearly they are a type of inductive step. Russell proposes that higher-level regularities can be used as indirect evidence for confirming or disconfirming more specific inductive generalizations.

If the above higher-level regularity is true, all Italians who John meets during the rest of his trip can speak Italian. The conjecture is reinforced when John meets another Italian who indeed speaks Italian.

Higher-level regularities, like any type of inductive generalization, are useful for making predictions about the world. After arriving at the hotel, John Smith approaches a receptionist. Even before she utters a single word, John already expects that she speaks Italian. This is a reasonable prediction to make since the receptionist is assumed to be Italian and John believes that all Italians speak Italian.

Knowing that the receptionist is Italian is sufficient to make a prediction about the language she speaks. More specific information, such as if she is from Northern Italy or Southern Italy, does not matter to the higher-level regularity. The receptionist can be from either place, and the prediction is still the same: she speaks Italian. On the contrary, if John discovers that she is a Greek temporary worker from Athens, no prediction about her language can be made using the higher-level regularity

Moreover, the prediction does not yield information any more specific than that the receptionist speaks Italian. For example, it is conceivable that there are several dialects within the Italian language. In that case, John Smith has no knowledge of which dialect she speaks except that it is Italian.

It is not clear if Russell intends higher-level regularities to be true *all* the time. If a higher-level regularity is true only *some* of the time, then there is one or more counter-example that violates this regularity. In the Italian example, ethnic minorities may exist in Italy who do not speak Italian. This case would make a higher-level regularity correspond to one common definition of the term "heuristic," namely, a piece of knowledge that improves performance on some task (here, valid prediction about the world) most of the time, but does not guarantee to do so [Feigenbaum and Feldman, 1963].

If a higher-level regularity is true less than half of the time, using it to make predictions would be worse than random guessing. In fact, if less than half of all Italians speak Italian, then John Smith actually makes more correct predictions by randomly guessing whether the next Italian speaks Italian than using the higher-level regularity. The particular percentage (50%) can vary depending on the number of languages in the world. The more the languages, the smaller the percentage.

To perform better than random guessing, a higher-level regularity in general must be true more than 50% of the time. In some sense, this is the minimum level of accuracy one expects from the definition of a heuristic. If a higher-level regularity actually holds 100% of the time, one may argue that it is no longer a heuristic but a law (under most definitions of this term). So, to be a heuristic, a higher-level regularity must be true somewhere between 50% and 100% of the time.

Suppose a higher-level regularity is true all the time. One implication is that there is no counter-example of this regularity in the world. Using the Italian example, an Italian who speaks no Italian simply is nonexistent in this world. Furthermore, we can distinguish something that *cannot* exist from something that *does not* exist in the world. In describing the physical world, for example, certain laws in science domains such as chemistry and biology hold true 100% of the time. One such law is "Acids reacts with Bases to form Alkalis." Exceptions to this law such as "Acids reacts with Bases to form Coca-Cola" simply cannot exist in the physical world. On the contrary, exceptions to rules describing a "softer" domain, such as Ledora's domain of terrorism, do not (rather than cannot) exist in the world. For example, imagine a world where the rule "If the supplier of arms is U.S.A., then the receiver is a non-communist country" is true all the time. Exceptions to this rule, for instance, U.S.A. supplying arms to China, *can* exist in the world (in principle) but do not in actuality.

2.2.8 Summary

The discovery systems discussed above - BACON, GLAUBER, STAHL, AM and Meta-DENDRAL - can all be viewed as performing a search process. The goal of each of these systems is to discover some regularities in its respective domain. Operators that generate new states in the search space are generalization or specialization rules that formulate regularities to account for the input data. For BACON, the forms or templates that a regularity may take are given to the system. BACON's task is to fill in the variables and coefficient parameters in the templates. The templates constitute a kind of domain model in that they define the possible states of the search space. In BACON, the data are used only to verify the hypotheses. The degree of fit with the data serve as an evaluation function of how good a state is. Hence, the main driver is the domain model while the data serves only as an evaluation function. For GLAUBER and STAHL, no model is available to the system for defining possible states in the search space. Instead, the observed data drive the induction process. GLAUBER generalizes individual items to classes (a kind of variabilization). STAHL, on the other hand, uses some domain-specific heuristics to transform its input data into regularities. Of the three systems, STAHL is most relevant to our research because it uses the knowledge it has acquired so far as an evaluation function that effectively guides subsequent search. Hence, the search process in STAHL becomes increasingly informed as time progresses.

Meta-DENDRAL, like BACON, is a model-driven system. Its half-order theory determines what the states in the search space look like. In addition, data in Meta-DENDRAL are used to verify the proposed rules, rather than to generate them.

Like STAHL and GLAUBER, AM's search space is driven by data, except that it generates its own data. What data are generated depends on a notion of "interestingness." This notion is encoded in a set of powerful heuristics. Some of these heuristics are very domain-specific (e.g., "explore the inverse of a mathematical function"); the rest are domain-independent (e.g., "explore the extreme cases of something"). Because interestingness is *not* a dynamic quality, AM cannot be seen as acquiring a model of the mathematics domain. Rather, interestingness can be viewed as an evaluation function that determines how good a state (some augmented collection of concepts) is.

2.3 Efficiently Representing the Search Space: The Version Space Method

In viewing learning by example as a search problem, alternative concept descriptions correspond to states in the search space. Given a positive or negative example of a concept, generalization and specialization operators move the learning system from one state to another state. However, there may be many possible ways to traverse the search space by applying different operators on the same state. In other words, there might be several ways to generalize (or specialize) a concept description to admit (or reject) the new example. Depending on the chosen state, the learning system may need to backtrack to an alternative state later if the chosen state cannot be adjusted in any way to be consistent with some current example. The possibility of backtracking requires the reexamination of some previous examples in order to guarantee consistency with the new state. Hence, it would seem that the learning system must store all the training examples in case they are needed later. The need for storing and reexamining all past training examples is a source of inefficiency. This motivated Mitchell [1982] to develop a method for concept learning that requires neither backtracking nor reexamining previous training data.

One way to avoid backtracking to a previous state and remembering all training examples is to maintain a list of all possible states (concept descriptions) that are consistent with the current training example. Thus, if there are five ways to generalize (or specialize) a particular concept description to admit (or reject) the current example, this method will add all five new states to some list. When a new training example arrives, each state in the list is checked for consistency with the example. States that are not consistent with the example are generalized or specialized; those that cannot be adjusted are deleted from the list. This method does not involve backtracking because states outside this list must have been refuted by past data and hence cannot be the goal state. Also, no reexamination of past examples is required since each state in this list is consistent with all past training examples. However, because this method essentially maintains all "open" states in the space, it is inefficient in terms of both storage and processing.

Mitchell's [1982] insight was that not all states must be explicitly represented. Rather, it suffices to represent only the boundaries of that area of the search space that is consistent with all previously seen examples. Given a new example, the learning system adjusts the boundaries accordingly. Thus, the boundaries move closer and closer together until they converge to a single element that is the correct concept description.

Mitchell termed the data structures for representing these boundaries a concept's *version space*, and the algorithm that updates these boundaries the *candidate elimination* *algorithm*. Details about version spaces and the candidate elimination algorithm are given in the next subsection. Then, we discuss LEX, a system that uses version spaces for concept learning. Then, we describe a method for generating training examples that optimally converges a version space.

2.3.1 Candidate Elimination and Version Spaces

The candidate elimination algorithm requires a language for describing the concepts and the instances that is partially ordered by generality. Figure 2.2 shows the partially-ordered values of the attribute Occupation-of-Victim in the domain of international terrorism.

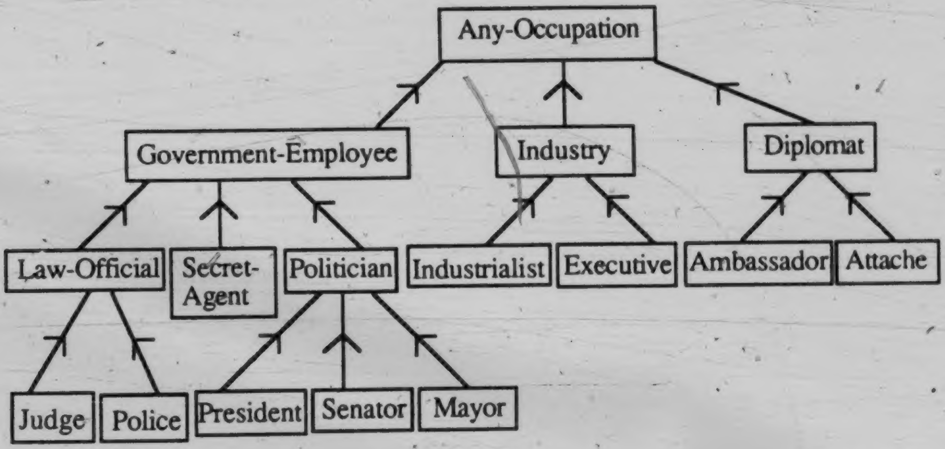

Any-Occupation
Government-Employee
Industry
Diplomat
Law-Official
Secret-Agent
Politician
Industrialist
Executive
Ambassador
Attache
Judge
Police
President
Senator
Mayor

Figure 2.2 Generalization Hierarchy for Occupation-of-Victim.

The states in the version space are plausible versions of a concept description. A description is considered plausible if it matches all known positive instances of the concept and none of the negative ones. A version space can be compactly represented using only two boundary sets: the maximally-general boundary set (G) and the maximally-specific boundary set (S). Together, G and S completely define a version space. Specifically, a description is a candidate version of a concept if (a) it is more specific than or equal to some version in *G*, and (b) it is more general than or equal to some version in *S*. Figure 2.3 describes in detail the candidate elimination algorithm in terms of how it computes and updates the *G* and *S* boundaries of a version space.

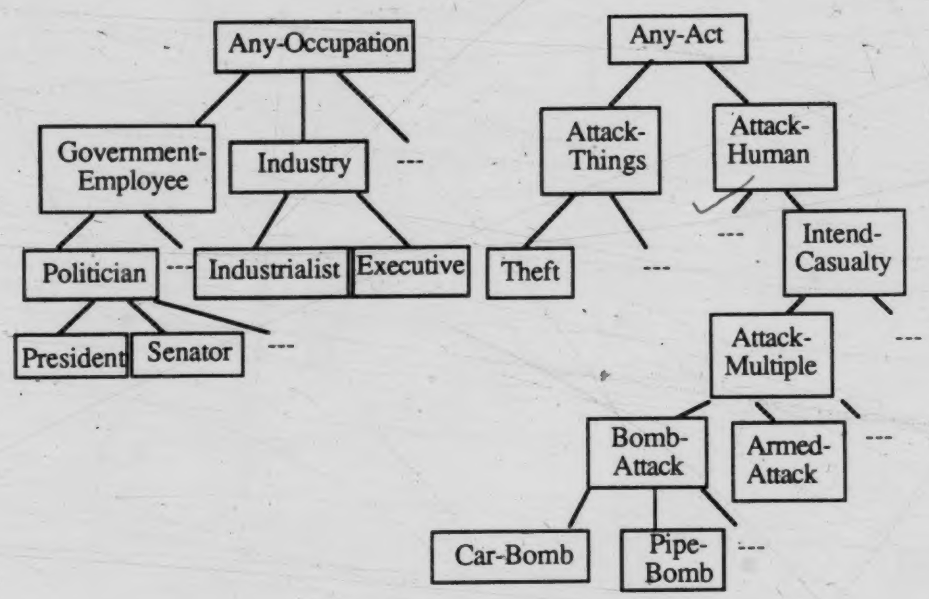
```
Initialize the sets S and G, respectively to the sets of maximally-specific and
maximally-general generalizations that are consistent with the first observed
positive training instance.

FOR EACH subsequent instance, i

  BEGIN
   IF i is a negative instance,

     BEGIN
      -Retain in S only those generalizations which do not match i.
      -Make generalizations in G that match i more specific, only to the extent
        required so that they no longer match i, and only in such ways that each
        remains more general than some generalization in S.
      -Remove from G any element that is more specific than some other
        element in G.
     END

   ELSE IF i is a positive instance,

     BEGIN
      -Retain in G only those generalizations that match i.
      -Generalize members of S that do not match i , only to the extent
        required to allow them to match i, and only in such ways that each
        remains more specific than some generalization in G.
      -Remove from S any element that is more general than some other
        element in S.
     END
  END
```

Figure 2.3 Candidate Elimination Algorithm.

Given a positive or negative instance of a concept, updating the corresponding version space can be viewed as a bi-directional search. A positive instance moves the *S* boundary toward *G*. The net effect is that the new *S* boundary is generalized to admit the positive instance. On the contrary, a negative instance specializes items in G, moving the G boundary toward S. The net effect is that no description within (more specific than) the G boundary matches the negative instance.

To illustrate how the candidate elimination algorithm works, we give an example below. Suppose that the desired concept description is *{Occupation-of-Victim:Politician Type-of-Activity:Attack-Multiple}*, and that the first positive instance of this concept is *{Occupation-of-Victim:President Type-of-Activity:Car-Bomb}*. G of the version space is initialized to be the singleton set *{Occupation-of-Victim:Any-Occupation Type-of-Activity:Any-Act}*. Note that Any-Occupation and Any-Act are the most general values for attributes Occupation-of-Victim and Type-of-Activity respectively. This makes sense because, without seeing any negative instance of the concept, the concept description can take on any value for the attributes Occupation-of-Victim and Type-of-Activity. On the other hand, S of the version space is initialized to be the first positive instance, i.e., *{Occupation-of-Victim:President Type-of-Activity:Car-Bomb}*. Since only one positive instance has been seen so far, the maximally-specific concept description is the instance itself. Figure 2.4a and 2.4b show the generalization hierarchies for Occupation-of-Victim and Type-of-Activity respectively. The version space after the first positive instance is as follows:

G: *{Occupation-of-Victim:Any-Occupation Type-of-Activity:Any-Act}*
S: *{Occupation-of-Victim:President Type-of-Activity:Car-Bomb}*


Any-Occupation
Government-Employee
Industry

Politician

Industrialist
Executive
President
Senator

Any-Act
Attack-Things
Attack-Human
Theft

Intend-Casualty
Attack-Multiple

Bomb-Attack
Armed-Attack

Car-Bomb
Pipe-Bomb
---

a) Occupation-of-Victim b) Type-of-Activity

Figure 2.4 Generalization Hierarchies of Two Attributes.

For each subsequent positive instance of the concept, each value in *S* that does not match (is not more general than or equal to) the corresponding value in the positive instance is generalized, only to the extent required to match it. For example, suppose the next positive instance is *{Occupation-of-Victim:Senator Type-of-Activity:Armed-Attack}*. The new *S* becomes *{Occupation-of-Victim:Politician Type-of-Activity:Attack-Multiple}*. Note that the most specific value that matches both President and Senator is Politician, and that the most specific value that matches both Car-Bomb and Armed-Attack is Attack-Multiple. The following is the version space after the above positive instance.

G: *{Occupation-of-Victim:Any-Occupation Type-of-Activity:Any-Act}*
S: *{Occupation-of-Victim:Politician Type-of-Activity:Attack-Multiple}*

For each negative instance of the concept, each value in G that matches, i.e., is more general than or the same as, the corresponding value in the negative instance is specialized, only to the extent required that they no longer match. Suppose G is currently *{Occupation-of-Victim:Any-Occupation Type-of-Activity:Attack-Human}*, and the next negative instance is *{Occupation-of-Victim:Industrialist Type-of-Activity:Armed-Attack}*. Note that *Occupation-of-Victim:Any-Occupation* and *Type-of-Activity:Attack-Human* match *Occupation-of-Victim:Industrialist* and *Type-of-Activity:Armed-Attack*, respectively. In this case, there are two ways to change G that are equally minimal: specialize attribute Occupation-of-Victim or specialize attribute Type-of-Activity. The resultant G boundary set contains two versions: *{Occupation-of-Victim:Any-Occupation Type-of-Activity:Bomb-Attack}* and *{Occupation-of-Victim:Government-Employee Type-of-Activity:Attack-Human}*. The first version in the new G corresponds to specializing Type-of-Activity minimally while holding Occupation-of-Victim the same; the second version corresponds to the opposite.

A concept is fully learned when G and S of the corresponding version space converge to a single description.

Success in using version spaces to learn concepts is independent of the order that the instances are presented. There are two major limitations to this approach of concept learning. First, it is designed to learn only conjunctive concepts but not disjunctive concepts. Second, the data is assumed to be noise-free, that is, instance descriptions are complete and correct, and are correctly classified as positive or negative.[3] In the following subsection, we review LEX, a learning system that uses the candidate elimination algorithm.

---

[3] Cohen and Feigenbaum [1982] discuss how the candidate elimination algorithm can be adapted to handle noisy data and acquire disjunctive concepts.

2.3.2 LEX

LEX [Mitchell, Utgoff & Banerji, 1982] is a learning system designed to acquire heuristics in the domain of symbolic integration. Initially, LEX is supplied with a set of operators for transforming mathematic expressions containing indefinite integrals. For example, Integration by Parts is one such operator and it is defined as $\int u\, dv \Rightarrow u.v - \int v\, du$. The left-hand side of the operator specifies situations where the operator *can* be validly applied, i.e., preconditions. LEX's goal is to learn heuristics for when the operators *should* be applied. The following is an example of a heuristic for applying Integration by Parts.

$\int$ x transc(x) dx => apply Integration by Parts
with u=x
and dv = transc(x) dx

It may be interpreted as "If the current problem contains an integrand which is the product of x and any transcendental function of x, then try Integration by Parts with u and dv bound to the indicated expressions."

The language for describing the objects in LEX - operators, heuristics, instances - can be defined in a grammar using a set of rewrite rules. A portion of this grammar is shown in Figure 2.5 in the form of a hierarchy.

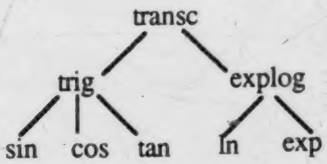


Figure 2.5 Some Rewrite Rules in LEX.

Learning heuristics in LEX corresponds to inductively generalizing from positive and negative examples of applying the corresponding operators. This learning process proceeds incrementally. Therefore, at any given time, heuristics are typically only partially-defined. Hence, LEX must define a suitable representation for partially-learned heuristics. In addition, LEX wants to bring all its current knowledge to bear on solving given integration problems, even knowledge about partially-learned heuristics. Thus, the representation of partially-learned heuristics must allow LEX to use whatever knowledge these heuristics have accumulated so far. Version spaces were chosen to be such representation.

LEX constructs its own practice problems, solves them using existing operators and heuristics, and then critiques each operator applied in its search for a solution as being either a positive or a negative instance for applying the operator. The task of the "practice-problem generator" in LEX is like that of a scientist. It designs experiments to collect new data for analysis later. Like a scientist, the problem generator designs the experiments without knowing in advance their results, in this case the heuristics. Instead, it relies on its current incomplete knowledge about heuristics to help create the practice problems.

The objective of the problem generator is to formulate *solvable* practice problems that will reduce the version space of existing heuristics. It first selects a partially-learned heuristic, and then generates a problem that matches some, but not all, versions in the version space of that heuristic. For example, given the following version space

$$\begin{aligned} G&: \int f_1(x) \quad f_2(x)\,dx \\ S&: \int 3x \quad \cos(x)\,dx \end{aligned}$$

the problem generator outputs the problem $\int 3x \sin(x)\,dx$, which is essentially S except that cos(x) is substituted by sin(x), its sibling in the hierarchy in Figure 2.5. Note that sin(x) matches $f_2(x)$ but not cos(x). The rationale for including a sibling of some term in S is that, by varying a known solvable problem in the smallest possible way, the problem generator hopes to generate another solvable problem. The major disadvantage of this approach is that the boundaries of the version space will move very slowly. The next section describes an instance-generation method that guarantees the optimal convergence of the version space.

2.3.3 Midpoint Method

The midpoint method is an instance generation algorithm that guarantees the quickest convergence of a version space assuming all concept descriptions in the version space are equally likely. By quickest convergence, we mean that the *G* and *S* boundaries of the version space will come to contain the same element after the smallest number of instances are seen. Basically, this method randomly selects an attribute *A* in the concept description, identifies *A*'s value in *G* as well as *A*'s value in *S*, and then finds the value that lies halfway between them in *A*'s generalization hierarchy. The resultant midpoint value becomes *A*'s value in the instance. The values of the rest of the attributes in the instance are obtained by copying the values of corresponding attributes in *S*. For example, suppose the version space of a particular concept in the terrorism domain is as follows:

G: *{Occupation-of-Victim:Any-Occupation Type-of-Activity:Any-Act}*
S: *{Occupation-of-Victim:Politician Type-of-Activity:Bomb-Attack}*.

Further suppose that Type-of-Activity is chosen as the attribute whose value is to be replaced. Then the instance generated using the midpoint method is *{Occupation-of-Victim:Politician Type-of-Activity:Intend-Casualty}*. Note that Intend-Casualty is located halfway between Any-Act and Bomb-Attack in the generalization hierarchy for Type-of-Activity. Regardless whether this is positive or negative, the above instance halves the version space of the attribute Type-of-Activity. If the instance is positive, then *S* becomes *{Occupation-of-Victim:Politician Type-of-Activity:Intend-Casualty}*. If the instance is negative, then *G* becomes *{Occupation-of-Victim:Any-Occupation Type-of-Activity:Attack-Multiple}*.

2.3.4 Summary of the Version Space Method

Version spaces and the candidate elimination algorithm are an efficient method for specifying and updating all versions of a concept description that are consistent with the instances of the concept seen so far. The source of knowledge used by the candidate elimination algorithm includes the partially-ordered language for describing concepts and instances, and the version spaces. Moreover, the midpoint method, using knowledge of this language and the version spaces, generates test instances that maximize the reduction in the version spaces.

2.4 Discussion of Abstracting and Using Domain Knowledge

Mitchell, Utgoff and Banerji [1982] raised the question of what *strategy* a learning system should use to learn multiple concepts simultaneously. Specifically, this question deals with two issues: (a) whether the learning system should concentrate on acquiring new concepts before refining existing ones, or postpone the refinement of concepts until a broad set of concepts are accumulated, and (b) which concept to learn first. As discussed above, the version space method uses knowledge about the concept/instance description language and the version spaces to acquire concept descriptions. However, to guide an *intelligent* search for concept descriptions, we need an additional source of knowledge besides the description language and the version spaces. Specifically, regularities abstracted in a domain can be used to account for observed data and to predict unobserved data. Together with the concept/instance language and the version spaces, these "higher-level" regularities (or domain model) constitute an important source of knowledge that can be used by a strategy for learning domain concepts.

This is the key idea behind Ledora: to include a discovery-like component that abstracts regularities about relationships among values of domain attributes *while* learning multiple concept descriptions. Ledora then verifies these regularities, with the goal of using them to accelerate the learning of the correct concept descriptions. Thus, the answer to "What strategy should be used in learning multiple concepts" is as follows. Examples across different concepts illustrate constraints in the domain. A learning system can identify and verify those constraints that are relevant to the concepts being learned, preferring constraints that will have the biggest impact on learning the concepts. Therefore, the question becomes not "What concepts should a learning system work on first?" but rather "How should a learning system co-ordinate learning the concept descriptions and learning about the domain in order to maximize learning of the concept descriptions?" In Chapter 3, we introduce the semantics of a particular form of domain model which we call predictive relations, and explain the method of abstracting these relations. Then, in Chapter 4, we discuss how these relations are used in Ledora to help co-ordinate activities in acquiring concept descriptions.

Chapter 3

Predictive Relations in a Structured Domain

This chapter presents the semantics of predictive relations for this work. As noted in Chapter 2, we view predictive relations as a form of higher-level regularities [Russell, 1986]. First, we describe Ledora's structured domain knowledge, and present the meaning of a predictive relation in the context of this type of knowledge. We then describe our method for detecting predictive relations from co-occurrence patterns of attribute-value pairs in instances of concepts that exist in the domain.

3.1 Structured Knowledge about Domain Attributes

Ledora operates in a "terrorist-activity" domain, consisting of attributes like Weapon, Occupation-of-Victim, Crime-Site, etc. Each attribute is represented as a collection of values that has a partial ordering of generality. These values are organized in a generalization hierarchy, such as the one shown in Figure 3.1 for the attribute Weapon. Each node represents a specialization subclass of its parent node. Thus, the "Gun" node represents, more precisely, Light-Weight-Weapons-that-are-Guns. Functionally, a generalization/specialization relation between two values is interpreted as a kind of asymmetric relation - anything that is a Submachine-Gun can also be described as a Gun, but the reverse does not hold. Thus, given a requirement that a Weapon be a Gun, any Weapon that is a descendant of Gun satisfies this requirement. Note that, in Figure 3.1, Light-Weapon is more general than Gun, but is neither more general nor more specific than Machine-Gun. We call the relationship between Light-Weapon and Machine-Gun *mutually exclusive* to denote that the two nodes have no children in common.

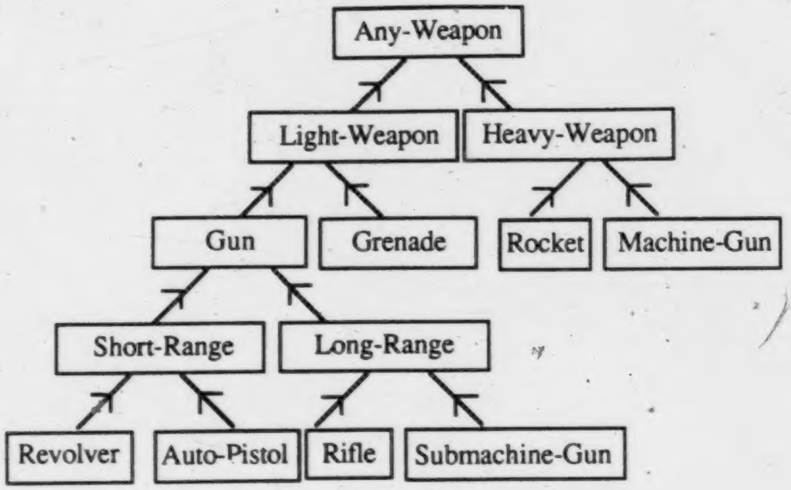

Any-Weapon
Light-Weapon
Heavy-Weapon
Gun
Grenade
Rocket
Machine-Gun
Short-Range
Long-Range
Revolver
Auto-Pistol
Rifle
Submachine-Gun

Figure 3.1 Generalization Hierarchy for the Weapon Attribute. (where each link is of type "Specialization-of")

3.2 Semantics of Predictive Relations

In the context of this type of domain knowledge, predictive relations have the form of a rule: *A:a* --> *B:b*, where *A* and *B* are attributes used to characterize instances of domain concepts, and *a* and *b* are values of the respective attributes. *A:a* is called the *predictor* in the relation and *B:b* the *predictable*. The meaning of a predictive relation is:

> "If attribute *A* and attribute *B* co-occur in some context *C* and attribute *A* has a value *a* in context *C*, then attribute *B* has the value *b* in context *C*."

For example, the predictive relation, ***Weapon:Revolver --> Occupation-of-Victim:President*** means that if the Weapon used (in a terrorist activity) is a Revolver, then the Victim is constrained to be a President. In the case of learning definitions for domain concepts, the "contexts" for attributes are concept instance descriptions.

In this study, predictive relations hold true in the domain 100% of the time. This means that all concept definitions and all instances encountered in the environment are consistent with the constraints placed on attribute-value co-occurrences that are implicit in predictive relations. In other words, if *A:a* -->*B:b* is true all the time, then instances with *A:a* and *B:b'*, where *b'* is neither *b* nor a particular value of *b*, *cannot exist* in the world. For example, given the predictive relation, *Weapon:Revolver* --> *Occupation-of-Victim:President,* instances with *Weapon:Revolver* and *Occupation-of-Victim:Industrialist* are nonexistent. This is because Industrialist is neither a President nor is it a specific type of President. Nonexistence means (a) two attribute-value pairs never co-occur in training instances presented by the environment, and (b) if Ledora were to construct an instance with these attribute-value pairs, the environment would classify it as nonexistent (e.g., "that just cannot happen"). The constraint that such instances are nonexistent is relevant for our use of the version space paradigm, because they can be treated as if they are negative instances: something that does not exist in the world cannot possibly be a positive instance of anything. We discuss the implications of detecting and using predictive relations that are true less than 100% of the time in the General Discussion.

3.3 Detecting Predictive Relations

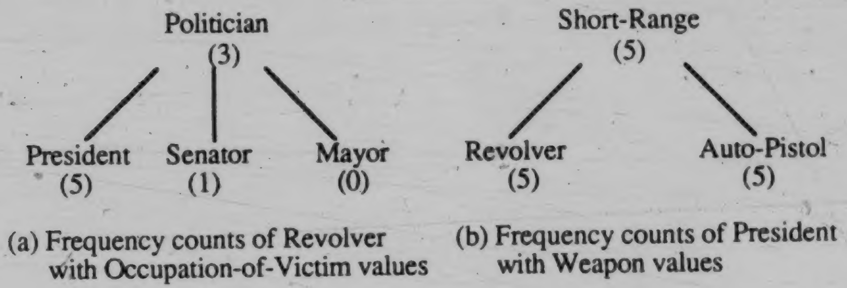
A predictive relation is a more constrained case of a co-occurrence relation. A co-occurrence between *A:a* and *B:b* means that *A:a* and *B:b* are either both present in an instance or both absent. The notation we use for a co-occurrence relation is *A:a* <--> *B:b*.

Predictive relations are discovered by checking for particular co-occurrence patterns of attribute-value pairs in positive instances of domain concepts. Specifically, a predictive relation of the form *A:a* --> *B:b* is defined to exist if: (a) *A:a* co-occurs with *B:b* more frequently than it does with any of *B:b*'s siblings, and (b) *B:b* does not co-occur more frequently with *A:a* than it does with any of *A:a*'s siblings. This second condition excludes the possibility that the relation is *B:b --> A:a* and the two conditions together exclude the case of a simple co-occurrence between the pairs.

The following example clarifies this method. Suppose the following instance sequence is observed:

1. *{Weapon:Revolver Occupation-of-Victim:President}*
2. *{Weapon:Revolver Occupation-of-Victim:President}*
3. *{Weapon:Grenade Occupation-of-Victim:Industrialist}*
4. *{Weapon:Revolver Occupation-of-Victim:President}*
5. *{Weapon:Revolver Occupation-of-Victim:President}*
6. *{Weapon:Revolver Occupation-of-Victim:President}*

These instances suggest the co-occurrence *Weapon:Revolver <--> Occupation-of-Victim:President*, because when *Weapon:Revolver* is present in instances, then *Occupation-of-Victim:President* is also present in these instances. The next step is to check whether *Weapon:Revolver --> Occupation-of-Victim:President* or *Occupation-of-Victim:President --> Weapon:Revolver* is true. Figure 3.2a gives some hypothetical frequency counts of Revolver co-occurring with President, its siblings, and its parent, Politician. Similarly, each value in Figure 3.2b represents the frequency count of President co-occurring with Revolver, its sibling Auto-Pistol, and its parent Short-Range. To test if the predictive relation *Weapon:Revolver --> Occupation-of-Victim:President* holds, Ledora first checks to see if the frequency count of *Weapon:Revolver* with *Occupation-of-Victim:President* (5 in Figure 3.2a) is greater than that of *Weapon:Revolver* with *Occupation-of-Victim:Senator* (1) or *Occupation-of-Victim:Mayor* (0). In this case, it is. Then, Ledora tests if *Occupation-of-Victim:President --> Weapon:Revolver* is true by reversing the above procedure. *Occupation-of-Victim:President* co-occurs with *Weapon:Auto-Pistol* as much as it does *Weapon:Revolver*, using the counts in Figure 3.2b. Thus, *Occupation-of-Victim:President* does not appear always to constrain the Weapon value to be Revolver. In other words, knowing that Occupation-of-Victim has value President does not allow us to unequivocally conclude that Weapon has the value Revolver. Thus, Ledora concludes that the proposed predictive relation is *Weapon:Revolver --> Occupation-of-Victim:President* and not *Occupation-of-Victim:President --> Weapon:Revolver*.



(a) Frequency counts of Revolver with Occupation-of-Victim values

(b) Frequency counts of President with Weapon values

Figure 3.2 Relevant values and frequency counts for testing.

In the above example, the proposed predictive relation involves only terminal node values from the attributes' generalization hierarchies. However, both the predictor and the predictable in a predictive relation may take subclasses as their values. Suppose Short-Range-Weapon is a class containing Revolver and Auto-Pistol, and Politician is a class containing Mayor, President, and Senator. The meaning of *Weapon:Short-Range --> Occupation-of-Victim:Politician* is this: if the Weapon used belongs to the class Short-Range, then the Occupation-of-Victim belongs to the class Politician. Effectively, this relation predicts that if the Weapon used is either Revolver or Auto-Pistol, then the Occupation-of-Victim must be one of Mayor, President, or Senator. Note that the relation does not specify which of Mayor, President, or Senator is the value of Occupation-of-Victim.

Predictive relations involving classes in the predictor and/or predictable can evolve in two ways. One way is by observing instances that include members of those classes. Consider the following sequence:

1. *{Weapon:Revolver Occupation-of-Victim:Mayor}*
2. *{Weapon:Auto-Pistol Occupation-of-Victim:President}*
3. *{Weapon:Revolver Occupation-of-Victim:Senator}*
4. *{Weapon:Auto-Pistol Occupation-of-Victim:Mayor}*

Although the value of Short-Range and Politician have not been explicitly referenced in the instance descriptions, this set is indirect evidence that a co-occurrence relation exists between *Weapon:Short-Range* and *Occupation-of-Victim:Politician.* We enable this recognition by boosting the frequency counts for non-terminal attribute-value pairs when their children co-occur. To show *Weapon:Short-Range --> Occupation-of-Victim:Politician* (or vice versa), the method used is the same as outlined previously. Short-Range must co-occur more frequently with Politician than it does with siblings of Politician (e.g., Law-Official and Secret-Agent). To disprove *Occupation-of-Victim:Politician --> Weapon:Short-Range*, the alternative set of frequency counts is checked, just as described above. The important point is that frequency information about attribute-value pairs does not depend on actually seeing those pairs explicitly co-occurring in an instance; it is accrued indirectly as well. The exact details of how this happens are given in the next chapter.

Another way of detecting predictive relations that involve subclasses is by generalizing existing relations. It is possible that, after the discovery of *Weapon:Revolver --> Occupation-of-Victim:President*, a related predictive relation such as *Weapon:Revolver --> Occupation-of-Victim:Senator* is also noticed. For example, both predictive relations are discovered in the following instance sequence:

1. *{Weapon:Revolver Occupation-of-Victim:President}*
2. *{Weapon:Revolver Occupation-of-Victim:President}*
3. *{Weapon:Revolver Occupation-of-Victim:President}*
4. *{Weapon:Revolver Occupation-of-Victim:Senator}*
5. *{Weapon:Revolver Occupation-of-Victim:Senator}*
6. *{Weapon:Revolver Occupation-of-Victim:Senator}*

These two predictive relations together suggest that the real relation might be *Weapon:Revolver --> Occupation-of-Victim:Politician.*

It is not always completely certain that an accurate generalization will be made when only some of the class members of the generalized predictor (or predictable) participate in specific predictive relations. In the above example, because Senator and President comprise only two of the three subclasses of Politician (instances with the third subclass, Mayor, have not been observed), it is not clear that the generalized predictive relation would be accurate. Indeed, the "reality" of the domain might be that there are simply two separate relations, *Weapon:Revolver --> Occupation-of-Victim:President* and *Weapon:Revolver --> Occupation-of-Victim:Senator*, excluding the possibility of predicting Mayor, given Revolver.

Nevertheless, by knowing the structure of domain values and the nature of current predictive relations, Ledora can conjecture that these kinds of generalized predictive relations might be worth investigating. It would be worth doing so to not only simplify the domain model (reduce the number of predictive relations), but also to increase its predictive power. One way to guide this process is really intuition: if a proposed predictive relation seems to involve most but not all the values in the same class, design some tests to establish whether the as-yet-unseen values in the class are also involved in the relation. If they are, then generalize. Whether or not conducting these tests is *pragmatic* depends on how many unseen values there are. Whether or not such tests are *worth* doing so, i.e., making the effort to investigate a generalized predictive relation, is still another matter. The next chapter, which presents the observe-theorize-experiment framework, describes how Ledora designs such tests and decides whether such tests are worth doing.

3.4 Summary

The important points to remember about a predictive relation, *A:a --> B:b*, are the following: (a) *A:a* is the predictor and it constrains the predictable attribute, *B*, to have value *b*; (b) Potential predictive relations are determined by particular co-occurrence frequency counts that are consistent with a "one-way" constraining relationship; (c) Frequency counts are propagated through the generalization hierarchy, enabling predictive relations that involve non-terminal values to be recognized; (d) Predictive relations can also be ***generalized***; (e) Predictive relations hold true 100% of the time in the domain we investigate; (f) If *A:a* --> *B:b* is true, then *A:a* cannot co-occur with $B{:}b'$ in any instance of any concept in the domain. Such instances are never generated by the environment as training instances for Ledora; (g) If *A:a* --> *B:b* is true and Ledora generates, for some concept, its own test instance that includes *A:a* and $B{:}b'$, the environment will classify the instance as "nonexistent." Points (f) and (g) follow from the constraint specified in (e).

Chapter 4

Learning Framework Design

4.1 Overview

This chapter describes the framework in which Ledora learns domain concept definitions and predictive relations. First, we explain how the domain is set up in terms of concept definitions, attributes, and predictive relations. We then present the representation of concepts and instances. Next, we describe the data structure and processes that maintain co-occurrence information about attribute-value pairs. These support the method for detecting predictive relations that was outlined in the previous chapter. Predictive relations also have associated confidence ratings, and we outline how new instances affect the confidence ratings of existing predictive relations.

Finally, we present Ledora's observe-theorize-experiment framework. At any time, Ledora is in one of these three modes. In observational mode, Ledora receives a positive or negative instance of a particular concept from the environment. As explained in Chapter 3, certain co-occurrence patterns between attribute-value pairs in these instances propose the creation of predictive relations. When this happens, Ledora enters theory mode to explore the credibility of the proposed predictive relations. If Ledora establishes the credibility of proposed predictive relations in theory mode, it uses these relations to generate test instances for partially-learned concepts, asking the environment to classify them. We call this activity experimental mode because Ledora runs an "experiment" of the form "Is this description a positive or negative instance of Concept C," using an abstracted and verified predictive relation as the basis of the experiment.

4.2 World Simulator: Defining the Domain

We implemented a program module called the world simulator that is separate from Ledora. The world simulator effectively sets up the domain with which Ledora interacts. Specifically, we defined to the world simulator (a) a set of domain attributes, where each attribute is defined by a collection of values that form a generalization hierarchy, (b) the number of target domain concepts that Ledora must learn, and for each concept, a target concept definition described using attribute-value pairs, and (c) a set of target predictive relations that hold true 100% of the time in this domain, i.e., all concept definitions and all instances are consistent with the constraints specified by each target predictive relation. Thus, given the target relation *A:a* --> *B:b*, any concept definition and any instance generated by the world simulator that contains *A:a* (or something more specific) must contain *B:b* (or something more specific). Unlike the world simulator, Ledora only has knowledge of (a) but not (b), or (c).

Figure 4.1 shows the interactions between Ledora and the world simulator. Case 1 corresponds to observational mode. Specifically, the world simulator generates, using the knowledge outlined above, classified concept instances that adhere to constraints set by the target predictive relations. Then the world simulator passes these instances to Ledora. In case 2, Ledora asks the world simulator to classify concept instances that it generates in theory and experimental modes. The world simulator checks each instance of a concept against the concept's definition, and returns the classification of the instance to Ledora.

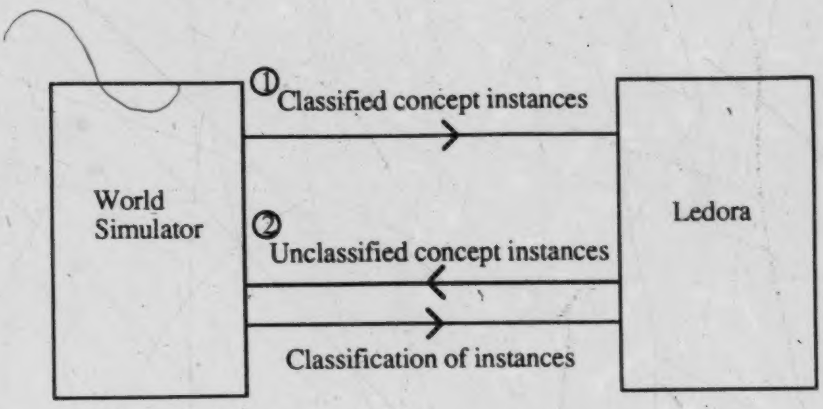

World Simulator
①Classified concept instances
②Unclassified concept instances
Classification of instances
Ledora

Figure 4.1 Interactions between World Simulator and Ledora.

4.3 Representation of Concepts and Instances

Both concept definitions and instances are described as sets of attribute-value pairs. For example, the concept Terrorism-in-South-America might be defined as *{Occupation-of-Victim:Politician Crime-Site:Street Weapon:Gun}*. An example of a concept instance gives the name of the concept, the type of instance (positive or negative), and the instance description, such as *{Terrorism-in-South-America positive Occupation-of-Victim:President Crime-Site:Street Weapon:Rifle}*. Partially-learned concepts are represented by version spaces [Mitchell, 1977] and updated by the candidate elimination algorithm presented in Chapter 2.

Each value in a positive instance of a concept must be either more specific than or the same as the corresponding value in the concept definition. The above instance description is a positive instance of Terrorism-in-South-America because President and Rifle are more specific values of Politician and Gun, respectively.

A negative instance of a concept must have at least one attribute whose value is either more general than or mutually-exclusive with the corresponding value in the concept. For example, *{Occupation-of-Victim:Industrialist Crime-Site:Street*

Weapon:Gun} is a negative instance of Terrorism-in-South-America because Industrialist is mutually-exclusive with Politician. Similarly, *{Occupation-of-Victim:Government-Employee Crime-Site:Street Weapon:Gun}* is a negative instance of Terrorism-in-South-America because Government-Employee is more general than Politician.

In this study, all instance descriptions, regardless of whether they are generated by Ledora or by the world simulator, contained only terminal values for attributes. This corresponds to domains in which events or instances can only "happen" with specific attribute values. For example, one may think of a chemistry experiment in terms of combining some type of acid with some type of alkali, but for the actual experiment event, a particular acid and alkali must be selected. Alternatively, it is also justifiable to allow instance descriptions to contain non-terminal values. This would correspond to high-level event descriptions that often occur in the world, such as "The assassin used an automatic weapon to kill a high-ranking politician." Further details on the type (i.e., subclass) of automatic weapon or the type of politician are not available in this description.

The Appendix gives the complete set of attributes and values for the terrorism domain used in this study.

4.4 Maintaining Co-occurrence Information

The previous chapter describes the frequency patterns that must hold among attribute-value pairs that propose a potential predictive relation. We use a data structure called a frequency table to maintain information about how frequently specific attribute-value pairs co-occur in instances.

Each entry in the frequency table consists of two attribute-value pairs, e.g., *{Weapon:Light-Weapon Occupation-of-Victim:Politician}*, and a frequency count indicating how often these two pairs have co-occurred in past instances. The frequency count is scaled between zero and one.

For each new instance, Ledora increments the corresponding frequency count for each possible pairing of values in the instance. Suppose the instance is *{Weapon:Long-Range Crime-Site:Street Occupation-of-Victim:Politician}*. The pairs to be incremented are *{Weapon:Long-Range Crime-Site:Street}*, *{Weapon:Long-Range Occupation-of-Victim:Politician}*, and *{Crime-Site:Street Occupation-of-Victim:Politician}*. Figures 4.2a and 4.2b present respectively a portion of the frequency table before and after this instance.

The frequency increment for a particular attribute-value pair is computed by multiplying the difference between the maximum possible frequency count (1.0) and the current count by a scaling factor. This factor specifies the size of the increment as a fraction of the difference from the frequency count to its upper limit. For example, suppose the scaling factor is 0.3, and *{Weapon:Long-Range Occupation-of-Victim:Politician}* has a frequency count of 0.2 before Ledora encounters the above instance. The frequency increment is therefore (1.0 - 0.2) x 0.3 = 0.24, giving a final frequency count of 0.44.

....	....	..
Weapon	Occupation-of-Victim	
Long-Range	Politician	.2
Light-Weapon	Govnt-Employee	.2
Light-Weapon	Politician	.2
Gun	Govnt-Employee	.2
Gun	Politician	.2
Long-Range	Govnt-Employee	.2
....	....	..
Weapon	Crime-Site	
Long-Range	Street	.2
....	....	
Crime-Site	Occupation-of-Victim	
Street	Politician	.2
....	....	..

....	....	..
Weapon	Occupation-of-Victim	
Long-Range	Politician	.44
Light-Weapon	Govnt-Employee	.37
Light-Weapon	Politician	.37
Gun	Govnt-Employee	.39
Gun	Politician	.39
Long-Range	Govnt-Employee	.39
....	....	..
Weapon	Crime-Site	
Long-Range	Street	.44
....	....	..
Crime-Site	Occupation-of-Victim	
Street	Politician	.44
....	....	..

a. Before the instance
{Weapon:Long-Range
Crime-Site:Street
Occupation-of-Victim:Politician}

b. After the instance.

Figure 4.2 Snapshots of the Frequency Table.

After incrementing the frequency count for each attribute-value pair that appears in the instance, Ledora increments the frequency counts for more general values associated with these pairs. We do this because the co-occurrence of the specific values in the instance is evidence, albeit indirect, that the more general values also co-occur. For example, after observing an instance that includes the pair *Occupation-of-Victim:Mayor* and *Type-of-Activity:Car-Bomb*, Ledora increases the frequency count associated with Politician (more general than Mayor) paired with Bomb-Attack (more general than Car-Bomb).[1] The amount of increment given to more general attribute-value pairs is a function of the number of levels between the more general value and the value that appears in the instance.

To propose a useful predictive relation between two attribute-value pairs, these pairs must first have co-occurred with some minimum frequency. For example, the frequency count for *{Weapon:Long-Range Occupation-of-Victim:Politician}*, currently at .44 in Figure 4.2b, must reach a threshold value (.5) before Ledora checks whether a proposed predictive relation exists between them, using the method presented in the previous chapter.

4.5 Monitoring Consistency of Predictive Relations

Each predictive relation has an associated confidence rating that indicates the degree to which previously seen concept instances are consistent with the relation. When a proposed predictive relation first emerges from the frequency table, its initial confidence is based on the frequency count of the predictor and predictable pair in the table. Although we might define only seven or eight target predictive relations in setting up the domain, many times that number emerge as proposed predictive relations, just on how attribute-value pairs *seem* to be predicting each other. Over time, new instances typically contradict these spurious predictive relations. Hence, Ledora must be able to discriminate spurious relations from relations that really hold true in the domain. To do this, Ledora continually adjusts a relation's confidence rating according to whether the relation is supported by incoming instances.

Once a predictive relation is proposed from the frequency co-occurrences, Ledora continually checks whether it is consistent with each new instance. Given an instance *I* and a proposed predictive relation *H* defined as *A:a --> B:b*, *H* is consistent with *I* if the following conditions are satisfied. First, *I* must contain both attributes *A* and *B*. Second, *A*'s value in *I* must be equal to or be more specific than the predictor. This means that *I* matches the predictor. Third, *B*'s value in *I* must be equal to or be more specific than the predictable. This fulfills *H*'s expectation for the value of attribute *B* in *I*, given that *A:a* occurs in *I*. For example, *Weapon:Long-Range --> Occupation-of-Victim:Politician* is consistent with the instance *{Weapon:Long-Range Occupation-of-Victim:Mayor Crime-Site:Street}*. This is because Mayor is more specific than Politician in the generalization hierarchy for Occupation-of-Victim.

---

[1] It also increments the pairs {Politician Car-Bomb} and {Mayor Bomb-Attack}. Basically, we generalize all possible pairings of seen values and each of their ancestors, up to some specified limit in the generalization hierarchy.

When a proposed predictive relation is consistent with a new instance, the relation's confidence rating is incremented. The size of a confidence increment depends on how close the predictor and the predictable values are to the corresponding values in the instance. Suppose predictive relations *H1* and *H2* are defined as *Weapon:Long-Range --> Occupation-of-Victim:Politician* and *Weapon:Long-Range --> Occupation-of-Victim:Government-Employee* respectively. They are both consistent with the instance *{Weapon:Long-Range Occupation-of-Victim:Mayor Crime-Site:Street}*. Mayor is a more specific value than both Politician and Government-Employee. However, *H1* is given a larger confidence increment because Mayor provides more evidence for Politician (the parent of Mayor) than for Government-Employee (a grandparent).

A proposed predictive relation *H* is inconsistent with an instance *I* if the following are true. First, *I* must contain both attributes *A* and *B*. Second, *A*'s value in *I* must be equal to or be more specific than the predictor. Again, this ensures that the instance matches the predictor. Third, *B*'s value in *I* must be more general than or be mutually-exclusive with the predictable. This makes *I* and *H* inconsistent because *I* does not meet *H*'s expectation for attribute *B*. For example, *Weapon:Long-Range -->*

Occupation-of-Victim:Politician is inconsistent with the instance *{Weapon:Long-Range Occupation-of-Victim:Executive Crime-Site:Street}*. This is because Executive and Politician are mutually-exclusive.

If a proposed predictive relation is inconsistent with an instance, the size of its confidence decrement does not depend on the position of the predictor and the predictable with respect to the corresponding values in the instance. For example, both *H1 (Weapon:Long-Range --> Occupation-of-Victim:Politician)* and *H2 (Weapon:Long-Range --> Occupation-of-Victim:Government-Employee)* above are inconsistent with the instance *{Weapon:Long-Range Occupation-of-Victim:Executive Crime-Site:Street}*. Note that Executive is mutually-exclusive with both Politician and Government-Employee. Although Government-Employee is a more general value than Politician, the confidence decrement to both *H1* and *H2* are the same because Executive is as much a counter-evidence to Politician as it is to Government-Employee.

A proposed predictive relation *H* may be neutral with respect to an instance *I* if it is neither consistent nor inconsistent with *I*. This can happen if *I* does not contain both attributes *A* and *B*, or *I* does not match the predictor. The latter case occurs if the value of *A* in *I* is more general than or mutually-exclusive with the predictor. For example, *Weapon:Long-Range --> Occupation-of-Victim:Politician* is neutral to the instance *{Weapon:Short-Range Occupation-of-Victim:Mayor Crime-Site:Street}*. This is because Long-Range and Short-Range are siblings, hence mutually-exclusive.

4.6 The Observe-Theorize-Experiment Framework

Ledora learns multiple concepts, with the aid of verified predictive relations, in an observe-theorize-experiment framework. These modes differ in the source of the instances and whether the instances are being used to learn domain concepts or to explore proposed predictive relations.

Figure 4.3 gives the general control structure of how Ledora's tasks are organized within these modes. Because Ledora does not know any predictive relation in the beginning, it cannot enter theory or experimental mode at that time. Therefore, Ledora always begins in observational mode, receiving classified instances, one at at time, for associated concepts. It moves to theory mode if predictive relations have been proposed that (a) have some minimum degree of confidence, (b) are useful to learning several concepts if they are verified, and (c) can generate instances that cause more convergence than instances generated with the midpoint method. If activities in theory mode verify a proposed predictive relation, Ledora immediately attempts to use this relation in experimental mode to generate test instances for partially-learned concepts. Given a choice between testing proposed predictive relations in theory mode and using verified predictive relations in experimental mode, Ledora chooses the latter. This is compatible with Ledora's main goal of learning concept definitions. Regardless of whether Ledora is in observational, theory, or experimental mode, it (a) updates version spaces and co-occurrence information for each instance it encounters, (b) verifies if existing predictive relations are consistent with the current instance, adjusting confidences as necessary, and (c) notices any new predictive relations and generalizes them if possible.

The next sections elaborate on important details of observational, theory, and experimental modes. In both theory and experimental modes, Ledora creates test instance of particular concepts for the world to classify, although to satisfy different goals. The most important aspects of theory mode concern how Ledora generates instances to verify a proposed predictive relation, and proposes and tests generalizations of existing predictive relations. For experimental mode, the key issues involve how Ledora determines that using a verified predictive relation causes greater version-space convergence than the midpoint method. Before describing the three learning modes in detail, we explain how Ledora first recognizes that a predictive relation is relevant to its partially-learned concepts.

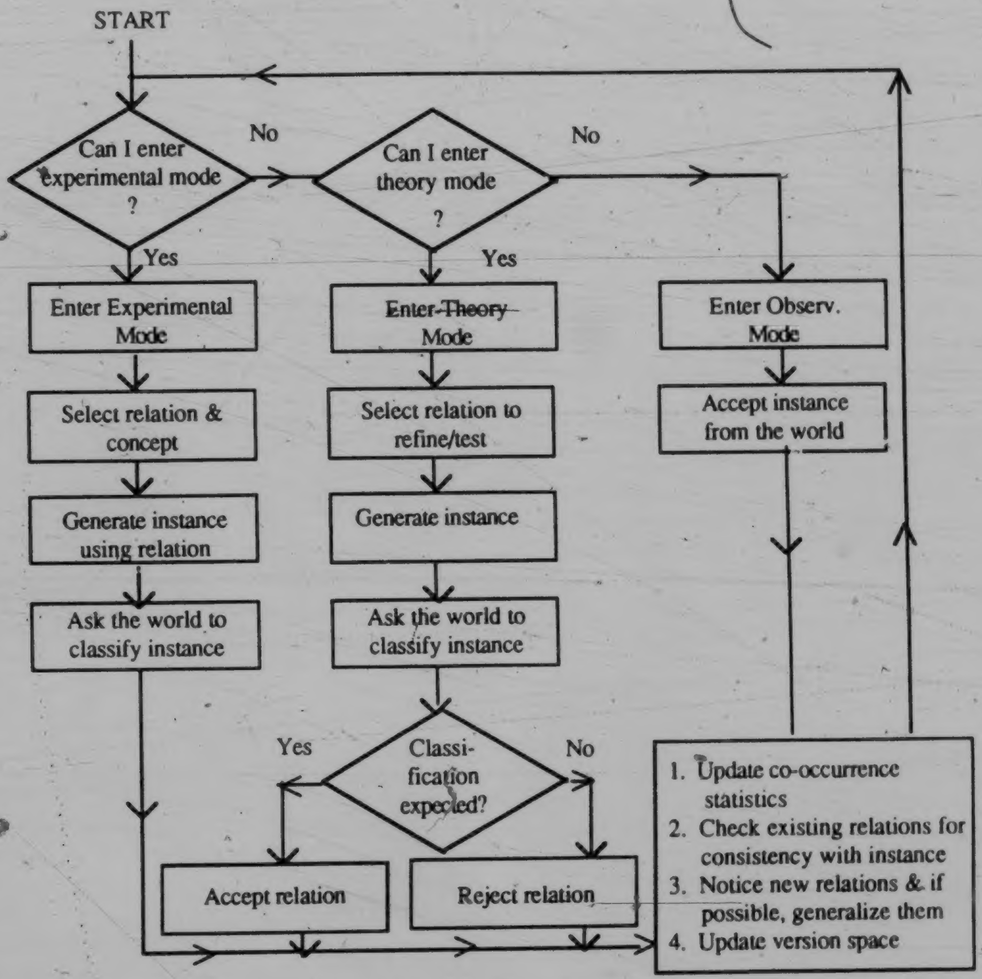

START
Can I enter experimental mode ?
No
Yes
Can I enter theory mode ?
No
Yes
Enter Experimental Mode
Enter Theory Mode
Enter Observ. Mode
Select relation & concept
Select relation to refine/test
Accept instance from the world
Generate instance using relation
Generate instance
Ask the world to classify instance
Ask the world to classify instance
Classification expected?
Yes
No
Accept relation
Reject relation
1. Update co-occurrence statistics
2. Check existing relations for consistency with instance
3. Notice new relations & if possible, generalize them
4. Update version space

Figure 4.3 Ledora's Control Structure.

4.6.1 The Applicability of Predictive Relations

In both theory mode and experimental mode, Ledora generates instance descriptions and asks the world to classify them. Obviously, an instance description must be defined with respect to some partially-learned concept. Although theory and experimental modes generate test instances for different purposes, in both cases a concept must be selected as a context for verifying or using the predictive relation.

Recall that a concept's version space is defined by a S boundary and a G boundary that specify respectively the most specific version and the most general version(s) of the concept definition that is consistent with all past instances. An *attribute* is *fully-converged* if the value of that attribute is the same in both the G and S boundaries. For instance, the attribute Crime-Site is fully-converged in the following version space.

G: {Crime-Site:Urban Type-of-Activity:Attack-Human}
S: {Crime-Site:Urban Type-of-Activity:Sniping}.

Note that there may be other attributes in the version space (Type-of-Activity, in this case) that are not fully-converged. A *version space* is *fully-converged* if all concept attributes are fully-converged, i.e., G and S are identical. For example, the following version space is fully-converged.

G: {Crime-Site:Urban Type-of-Activity:Sniping}
S: {Crime-Site:Urban Type-of-Activity:Sniping}.

We call a predictive relation *applicable to a concept* if that relation can be used to generate a test instance that would reduce the concept's version space. A predictive relation $A{:}a \rightarrow B{:}b$ is applicable to a concept if the following are satisfied. First, the concept definition must include both attributes A and B. If it doesn't, then the predictive relation has no bearing on the definition. Second, the predictable's value must be equal to or lie within the G and S boundaries for that attribute in the concept's current version space. Third, the predictor's value must be more general than or be equal to the corresponding value in *S*. We discuss these conditions in more detail below.

4.6.1.1 Applicability Constraints on the Predictable

For a predictive relation *A:a-->B:b* to be applicable to concept *C*, the value of the predictable must lie within *C*'s current version space. In other words, the predictable must be equal to the corresponding value in either *G* or *S* or be somewhere in between. If the predictable's value lies outside the version space, then clearly it is not part of the concept definition and any domain relation that constrains its value will not provide any new information about that particular concept's definition. For example, the predictive relation *Weapon:Long-Range --> Type-of-Activity:Attack-Human* is *not* applicable to the concept defined by the following version space:

G: {Weapon:Long-Range Type-of-Activity:Intend-Casualty}
S: {Weapon:Rifle Type-of-Activity:Sniping}.

Because the predictable (*Type-of-Activity:Attack-Human*) is more general than the corresponding value in *G* (*Type-of-Activity:Intend-Casualty*), the value for Type-of-Activity in the version space must converge to something more specific than the predictable's value. Thus, using the relation *Weapon:Long-Range --> Type-of-Activity:Attack-Human* to generate an instance that includes *Type-of-Activity:Attack-Human* is not helpful for learning the concept.

4.6.1.2 Applicability Constraints on the Predictor

A predictive relation of the form *A:a --> B:b* can be used to make a conclusion about the value of feature *B*, provided that the context in which this relation is used matches the predictor. In this learning task, the partially-learned definition of a concept is the context. Therefore, this partial definition must match the predictor in order to make a valid conclusion about the predictable's value.

For example, suppose we know a predictive relation such as *If the site of the terrorist activity is a place of Entertainment, then the terrorist activity is an Assassination*. We then encounter some description of an event in the world, like "Terrorists attacked in a Bar in West Germany." The Crime-Site value in this event, Bar, is a specialization of Entertainment-Place. Hence, it matches the predictor and so we conclude that the activity is an Assassination.

Suppose that the above event occurred in a "Public-Place." Is it still valid to conclude that the activity is an Assassination? No, because while a Public-Place can indeed be a place of Entertainment, it can also be an Airport, a Hotel, or some other specialized Public-Place.

Thus, Ledora must establish whether the current concept definition matches the predictor's value in order to use the predictive relation. However, it wants to use these domain relations when the concept definition is only partially-learned. This means that the concept definition may or may not ultimately converge to a value that matches the predictor's value. This problem of deciding whether a predictor is matched by a partially-learned concept can be reduced to considering where the predictor's value lies with respect to the G and S boundaries. Regardless of how the version space converges, there are situations in which Ledora knows with certainty whether or not the final concept definition must match the predictor. For other cases, it is uncertain whether the final concept definition will match the predictor. In these situations, Ledora ranks the degree of certainty by considering the placement of the predictor with respect to G and S. We explain these cases in the following subsections.

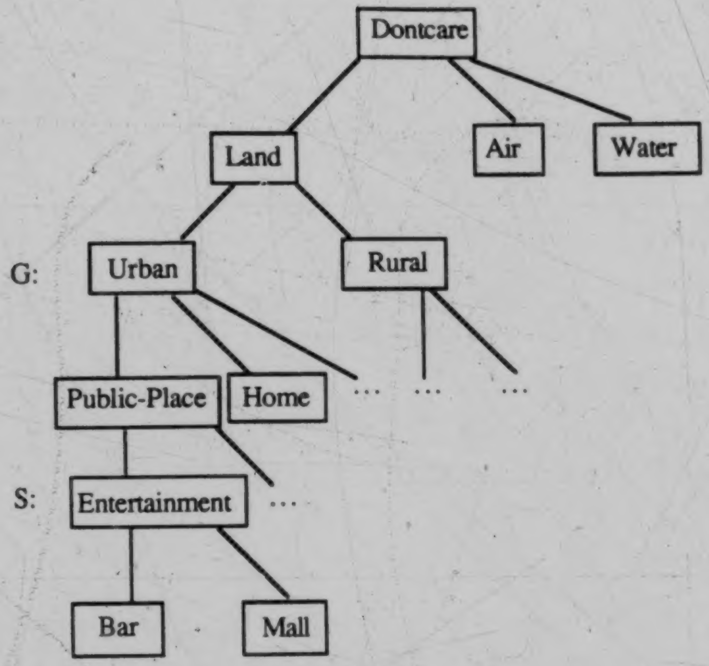


Figure 4.4 Partial Generalization Hierarchy for Crime-Site.

A guaranteed match to the predictor. The ideal case is when attribute *A* in *C*'s version space has already converged to the predictor's value i.e., *A:a*. In this case, the concept definition definitely includes *A:a* and it matches any predictive relation in which *A:a* occurs as predictor.

Suppose that attribute *A* is not fully converged and that the predictor is more general than or equal to the corresponding value in *G*. In this case, *C* matches the predictor because attribute *A* *must* converge to some value between *G* and *S* inclusive and they are all more specific than the predictor. Consider the generalization hierarchy for Crime-site given in Figure 4.4 and a concept *C* represented by the following version space:

G: {Crime-Site:Urban Type-of-Activity:Attack-Human}
S: {Crime-Site:Entertainment-Place Type-of-Activity:Sniping}.

Suppose the predictive relation is *Crime-Site:Land --> Type-of-Activity:Assassination*. This relation says that *If the crime takes place on Land, then the activity involves some kind of Assassination*. Because Land is more general than Urban, which is the most general value to which *C* can possibly converge, *C* matches the predictor.

Similarly, if the predictor's value is equal to the value in *G*, then attribute *A* can only converge to some value more specific than or the same as the value in *G*. For example, the predictive relation *Crime-Site:Urban --> Type-of-Activity:Assassination* means *If the crime occurs in an Urban place, then the activity involves some kind of Assassination*. Because the concept can only converge to Urban or some value more specific, it matches the relation's predictor.

A guaranteed mismatch to the predictor. There are three cases where Ledora can be certain that the final definition of a partially-learned concept *C* does not match the predictor. The first is when the predictor's value is mutually-exclusive with the corresponding value in *G*. This means that they have no children in common, hence the predictor cannot be matched by any value to which the version space converges. For example, suppose *H* is *Crime-Site:Rural --> Type-of-Activity:Assassination*. Figure 4.4 shows that Rural and Urban have no children in common. Therefore, regardless of which value the attribute Crime-Site has in the final definition, it cannot match the predictor.

In the second case, the predictor is more specific than the corresponding value in *G* and mutually-exclusive with *S*. Using the above version space, an example of such a predictor is *Crime-Site:Home*. Attribute *A* can only converge to *G* or *S* or to a value more general than *S* but more specific than *G*. If *A* converges to *G*, it is more general than the predictor. Hence, it cannot match the predictor. If *A* converges to *S*, it too cannot match the predictor because the predictor is mutually-exclusive with *S*.

Similarly, if *A* converges to a value more general than *S* (but more specific than *G*), it cannot match the predictor. This is because the predictor and the corresponding value in *S* have no children in common, so any value more general than *S* must be more general than or be mutually-exclusive with the predictor. Suppose *H* is defined as *Crime-Site:Home --> Type-of-Activity:Assassination*. If Crime-Site actually converges to Urban, it is more general than Home. If it instead converges to either Public-Place or Entertainment-Place, it is mutually-exclusive with Home. Hence, regardless of what Crime-Site eventually is, it does not match the predictor.

In the final case of a mismatch, the predictor is more specific than the corresponding value in *S*. In this case, attribute *A* must converge to a value that is more general than the predictor. For example, suppose the predictive relation is *Crime-Site:Bar --> Type-of-Activity:Assassination*. The Crime-Site attribute can converge to either Urban, Public-Place or Entertainment-Place. They are all more general than Bar and hence can be described by a number of specializations, such as Mall. Hence, this partial definition for C does not match the constraint that Crime-Site must be a Bar.

Uncertain matches to the predictor. The above scenarios correspond to clear-cut cases of when a concept does and does not match the value of a predictor. There are two additional cases that represent uncertain matches. Suppose that the predictor is more general than *S* but more specific than *G*. Therefore, attribute *A* of the concept may converge to a value that is either more general than, the same as, or more specific than the predictor value. This is an uncertain match because the concept matches the predictor only if attribute *A*'s value turns out to be the same as or more specific than the predictor value. If *A*'s final value is more general than the predictor value, the concept does not match the predictor.

For this case, Ledora rates a predictive relation's match according to how close the predictor is to the concept's *G* boundary. The closer the predictor is to *G*, the higher the predictive relation's rated match. This is because the chance that *A* actually converges to a value more specific or equal to the predictor is better if the predictor lies very close to G[2]. Suppose *H* is defined as *Crime-Site:Public-Place --> Type-of-Activity:Assassination*. This relation means *If the crime occurs in an Urban place, then the activity involves some kind of Assassination*. Since Public-Place is halfway between *G* and *S*, the chance of Crime-Site converging to Public-Place or some value more specific is about half. Therefore, the match ranking in this case is medium.

The extreme version of this situation is when the predictor's value appears in the concept's *S* boundary. If attribute *A* converges to any value in the version space except *S*, its value is more general than the predictor. Hence, the chance of *C* matching the predictor is very small and as a consequence, *C*'s ranked match to *H* is given the lowest rating.

In summary, we rate the match of a predictive relation to a partially-learned concept by considering the relative position of the predictor to the concept's *G* and *S* boundaries. Ledora considers this rating, among other factors, when deciding which proposed predictive relations to test in theory mode or which verified predictive relations to use in experimental mode. Having described how Ledora determines the relevance of predictive relations to partially-learned concepts, we can now present how these relations are used in the observe-theorize-experiment learning framework.

4.6.2 Observational Mode

In observational mode, Ledora receives from the world, one at a time, randomly-generated instances of concepts with associated classifications. First, a concept is selected from the set of target concepts.[3] The decision to present a positive or negative instance of this concept is decided randomly. An instance description is then generated by randomly selecting the values for the required attributes, within the constraints that it (a) meets (violates) the concept definition if the instance is positive (negative), and (b) is consistent with each of the target domain predictive relations. Thus, the target relations that hold true in the world plus the particular concept definition are the constraints on an otherwise random process of instance generation.

---

[2] Thinking of this spatially, the predictor divides the version space into two subspaces: *S* to predictor, and predictor to *G*. If the first subspace (*S* to predictor) is large, the probability for that concept attribute to converge within that space is also correspondingly large.

Ledora does not know, at any given time, how many different concepts it is required to learn. It simply creates a version space for a new concept when it first encounters a positive instance of that new concept.

4.6.3 Theory Mode

Theory mode performs two kinds of tasks. One kind of task checks the credibility of proposed predictive relations that have been abstracted from the frequency information. Another kind of task tests whether new predictive relations should be proposed by generalizing the predictor and/or the predictable of existing predictive relations. Theory mode uses a *task queue* to order all the above tasks according to priority. All verification tasks have a higher priority in the task queue than all generalization tasks. This is because verified predictive relations are used in experimental mode for learning concept definitions, the primary task of Ledora.

4.6.3.1 Testing Prediction Credibility

When a predictive relation $A{:}a \dashrightarrow B{:}b$ is proposed from the co-occurrence patterns, it is based on the fact that Ledora has seen instances containing $A{:}a$ and $B{:}b$, and has not seen instances containing $A{:}a$ and $B{:}b'$, where b' is a sibling of b. In other words, it may be due to chance that instances with *A:a* and *B:b'* have not yet appeared. It is the aim of theory mode to establish the credibility of proposed predictive relations by actively testing this condition. The validity checks depend on the formal definition of predictive relations, namely that they represent a constraint that holds 100% of the time in the domain. Thus, the predictive relation *A:a* --> *B:b* means that *A:a* cannot co-occur with *B:b'*.

---

[3] The random or non-random choice of concept order was subsequently varied as an independent variable in our experiments.

This verification process is aimed at minimizing the cost of formulating predictive relations based solely on statistical evidence, i.e., the instances seen to date. If predictive relations are to be detected and used to advantage, some steps must be taken to sidestep the "wait for statistical evidence to accumulate" problem by testing directly the implication of predictive relations. Some pilot versions of Ledora indicated that, unless some verification like this is done, spurious predictive relations, which seem to have a good statistical track record, may lead to bad "experiments" in experimental mode.

To focus the effort in theory mode, Ledora only tests proposed predictive relations that have some minimum degree of confidence and are useful in learning several domain concepts. This ensures a reasonable payoff in learning domain concept definitions for the effort of establishing that a proposed predictive relation is credible. In addition, Ledora only verifies proposed predictive relations that can be used to generate a test instance that reduces some concept's version space by more than 50%. This guarantees that using the verified predictive relations has some advantages over the midpoint method. We discuss this in more detail in a later section.

For each proposed relation that satisfies all the above conditions, Ledora creates a verification task on the task queue, where each verification task description indicates (a) the number of domain concepts to which the relation is applicable, and (b) the number of such concepts whose version spaces are converged more than 50% by the relation. Ledora eventually executes all verification tasks on the queue, starting with the most promising one.[4] The most promising verification task is the one involving a relation that is applicable to the largest number of concepts on which the relation has the greatest impact.

To verify a particular proposed predictive relation, Ledora generates a test instance that, if the world claims is nonexistent, serves as evidence that the predictive relation is credible. Because this instance must be created with respect to some concept, Ledora first chooses a concept that includes both attributes *A* and *B* in its definition and whose current definition matches the predictor (as explained in section 4.6.1.2).

After choosing a concept as a context for testing *A:a* --> *B:b*, Ledora creates an instance of that concept by copying the values for all attributes, except *B*, from the concept's *S* boundary. Attribute *B* of the instance is given a value b', which is a sibling of *b*. If any value in the resultant instance corresponds to a non-terminal node in a generalization hierarchy, then that value is replaced by one of its randomly-chosen terminal descendents. The justification is that, because the terminal value is more specific, it "matches" the more general non-terminal value. Therefore, in testing a predictive relation, the expectation for instances with non-terminal values (e.g., Politician) must also hold for instances with terminal values (e.g., Senator). Hence, both tests using the non-terminal values and tests using the terminal values will produce the same conclusions about the validity of a predictive relation.

An instance generated using the above method is a "near-miss" because it differs from a known positive instance (i.e., the *S* boundary) by only one attribute, namely *B*. If the predictive relation *A:a* --> *B:b* is true, Ledora expects this instance description, which contains *A:a* (or something more specific) and *B:b′* (or something more specific), to be classified as nonexistent.

---

[4] As noted in the control structure, any verified predictive relation will cause Ledora to enter experimental mode immediately. After finishing the experiment tasks, control will then return to theory mode to continue verification tasks. Hence, control moves back and forth between theory and experimental modes until all theory tasks are finished, at which point it returns to observational mode.

The following example clarifies this. Suppose that a concept has the version space:

G: {Weapon:Any-Weapon Occupation-of-Victim:Any-Occupation Crime-Site:Land}
S: {Weapon:Rifle Occupation-of-Victim:Mayor Crime-Site:Street}

and that there is a proposed predictive relation *Weapon:Long-Range --> Occupation-of-Victim:Politician*. Ledora generates an instance by copying from *S Weapon:Rifle* and *Crime-Site:Street*, which are both terminal values. The predictable attribute Occupation-of-Victim is instantiated with a sibling of Politician, say Law-Official. Because Law-Official is non-terminal, it is replaced by Judge, one of its terminal descendents. Thus, the final instance is *{Weapon:Rifle Occupation-of-Victim:Judge Crime-Site:Street}*.

Ledora then asks the world to classify this instance. If the predictive relation *Weapon:Long-Range --> Occupation-of-Victim:Politician* is true, then Ledora expects instances containing *Weapon:Long-Range* and *Occupation-of-Victim:Law-Official* (a sibling of Politician) to be nonexistent. Because Rifle and Judge are more specific values of Long-Range and Law-Official respectively, Ledora also expects the instance containing Rifle and Judge to be nonexistent. Therefore, if the world classifies this instance as being nonexistent, the proposed predictive relation has been verified. However, if the expectation is not met, i.e., the instance is positive or negative but not nonexistent, then Ledora discards the predictive relation.

In addition to discarding the discredited relation, Ledora also discards any other relations that are also implicated as being false. These relations have a predictor that is more general than or equal to the corresponding value in the test instance, and a predictable that is a sibling of the corresponding value in that test instance, i.e., a sibling of b'. The rationale for this is as follows: suppose that the values b_1, b_2, and b_3 are sibling values for some attribute *B*, and that there is a proposed relation $A{:}a$ --> $B{:}b_1$. An instance containing $A{:}a$ and $B{:}b_2$ that exists in the world (i.e., is not classified as nonexistent) is evidence against the credibility of $A{:}a$ --> $B{:}b_1$. Indirectly, this also serves as evidence against relations of the form $A{:}a$ --> $B{:}b_3$, because this relation suggests that $A{:}a$ cannot co-occur with $B{:}b_2$ either.

As noted earlier, this verification process rests on the assumption that predictive relations are true 100% of the time. Thus, if the predictor is matched, the predictable's value, and only that value, may exist in the world. Because all instances generated by the world adhere to this, this is a reasonable verification step. If, however, predictive relations hold less than 100% of the time, it is difficult to avoid the problem of just waiting for "enough" statistical evidence to build up. The implications of this are discussed in the General Discussion.

4.6.3.2 Generalizing Predictive Relations

Predictive relations may have related predictors and/or predictables such as *Type-of-Activity:Assassination --> Occupation-of-Victim:Senator* and *Type-of-Activity:Assassination --> Occupation-of-Victim:President*. The real predictive relation at work here might be *Type-of-Activity:Assassination --> Occupation-of-Victim:Politician*. Exploring this possibility is the second task of theory mode.

There are two situations in which Ledora generalizes predictive relations. The first case is when, for some value V_1, *all* its immediate specializations V_{11}, V_{12}, ... V_{1n} appear as the predictable (predictor) in predictive relations that have the same predictor (predictable). A new predictive relation is proposed with the same predictor (predictable) as these specific predictive relations and with a predicable (predictor) that is value V_1. We call this case *exclusive-sibling generalization*. For example, the predictive relation *Weapon: Long-Range --> Occupation of Victim:Politician* is proposed as a result of generalizing

Weapon:Long-Range --> Occupation-of-Victim:Mayor,
Weapon:Long-Range --> Occupation-of-Victim:President, and
Weapon:Long-Range --> Occupation-of-Victim:Senator,

where Mayor, President, and Senator are the only immediate specializations of the value Politician. This generalization step is not really inductive, in the sense that all the specializations have been seen and Ledora just rewrites the predictive relations in some equivalent form. However, it might be an inductive step if the system's knowledge base is incomplete, i.e., there are other specializations of Politician in the world, say Premier, that are currently unknown to Ledora.

It is overly restrictive to constrain generalization to this situation alone, because it entails waiting until enough evidence is accrued to propose predictive relations for all the specializations. Suppose, for example, that the third relation above, *Weapon:Long-Range --> Occupation-of-Victim:Senator*, were missing. A possible reason is that instances involving these values just haven't been seen yet by chance. Nevertheless, the higher-level generalization involving Politician, if true, may be very useful immediately. This is the second case where Ledora generalizes relations, and we call this case *non-exclusive sibling generalization*.

Generalization tasks in the task queue are ordered by (a) the percentage of the immediate specializations of the generalized relation's predictor (predictable) that support the generalization, and (b) the potential confidence of the generalized relation, as determined by the co-occurrence frequency of the corresponding predictor-predictable pair in the frequency table. To qualify as a generalization task, a possible generalization must be supported by some minimum percentage of all its specializations, and must have a minimum confidence. The generalization task with the highest priority in the queue is the one with the highest percentage of supporting immediate specializations and the highest potential confidence for the generalized relation.

Executing a generalization task means actively verifying the credibility of the proposed generalized relation by generating concept instances using that relation. Suppose that the proposed generalization is $A{:}a$ --> $B{:}b$. To test it, Ledora generates a *series of three instances* for a partially-learned concept that matches the predictor and includes the predictable attribute. Instance 1 of this series contains $A{:}a$ and $B{:}b$. If $A{:}a$ ($B{:}b$) is a non-terminal value, then it is replaced by one of its terminal descendents. Ledora expects this first instance to exist (i.e., be positive or negative, but not nonexistent) because if the proposed predictive relation is true, then $A{:}a$ and $B{:}b$ must at least co-occur. Instance 2 of the series contains $A{:}a$ (or a terminal descendent of $A{:}a$) and $B{:}b'$ (or a terminal descendent of $B{:}b'$). If the proposed predictive relation is valid, then $A{:}a$ cannot co-occur with $B{:}b'$, so Ledora expects this to be a nonexistent instance. Instance 3 of the series contains $A{:}a'$ (or a terminal descendent of $A{:}a'$) and $B{:}b$ (or a terminal descendent of $B{:}b$). If this instance exists, it indicates that $B{:}b$ can co-occur with something other than $A{:}a$. This verifies that the predictive relation is $A{:}a$ --> $B{:}b$, and not $B{:}b$ --> $A{:}a$.

If any of these expectations are not met, then Ledora does not create the generalized relation. It does, however, keep the more specific relations that initially suggested the generalization task.

In summary, theory mode is aimed at verifying proposed predictive relations and creating new ones. Instance generation and testing here is done strictly to serve these goals, although it requires selecting specific concepts as the "test beds" for these tasks. As noted earlier, version spaces and frequency counts are also updated in theory mode. However, the method of instance generation is aimed at testing proposed predictive relations, not at causing maximum convergence of partially-learned concepts. This latter goal is pursued in experimental mode.

4.6.4 Experimental Mode

We view acting as an experimenter as focusing on something to be discovered (here, some particular concept definition) and using whatever knowledge is available to guide that discovery. The midpoint method of instance generation uses knowledge of domain structure (attribute generalization hierarchies) and of version space structure. Verified predictive relations are an additional type of domain knowledge. Instance generation in experimental mode uses all three types of knowledge - attribute generalization hierarchies, version space structures, and verified predictive relations - to generate test instances that lead to maximal convergence of a partially-learned concept.

An instance generated using a verified predictive relation may not always help the version space for any particular concept to converge. The relation only specifies a constraint between two attribute-value pairs; it says nothing about whether an instance containing these is positive or negative for some concept. In contrast, the midpoint method guarantees reducing the version space of a concept by 50% (on that attribute), regardless of the classification. Thus, a verified predictive relation is valuable for generating instances only if Ledora can ensure that using it will be better than the midpoint method.

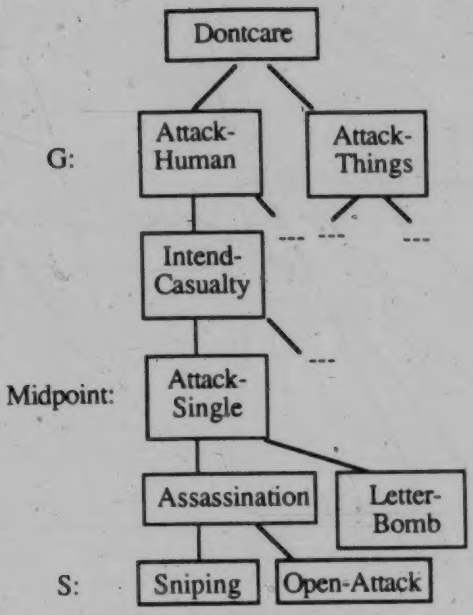
Ledora generates an experiment whenever there is at least one predictive relation *H* that satisfies the following conditions: (a) *H* must have some minimum degree of confidence and must have been tested in theory mode; (b) there must be some *C* to which *H* is applicable, i.e., *H* can be used to generate a test instance for *C*; and (c) the instance generated by using *H* must converge the attribute *B* in the version space by more than 50%. Experiments are inserted into a task queue to wait for their turn for execution. A priority rating for each experiment task is computed by an evaluation function incorporating *H*'s confidence, the degree of match of *C* to *H's* predictor, and the amount of convergence in *C*'s version space. In the next subsection, we detail how Ledora carries out an experiment. Then, we outline the condition under which using a verified predictive relation will guarantee a more informative test instance than using the midpoint method.

4.6.4.1 Running the Experiment

Suppose the verified predictive relation *H*, *A:a --> B:b* satisfies all the above conditions and is applicable to the concept *C*. Ledora generates a test instance for concept *C* by copying the values of all attributes, except *B*, from *S* of the concept's version space. If any of these values is non-terminal, Ledora substitutes that value by one of its terminal descendents. Attribute *B* of the instance is given the predicted value *b*. If *b* is a non-terminal value, then it is replaced by b_1, a terminal descendent of *b* such that *b* is the most specific generalization of b_1 and *B*'s value in *S*. This is justified because, regardless of its classification, an instance containing b_1 has the same effect on *G* (or *S*) as an instance containing *b*. For example, suppose *H* is *Weapon:Light-Weapon --> Type-of-Activity:Assassination* and *S* is *{Weapon:Rifle Type-of-Activity:Sniping Occupation-of-Victim:Senator}*. The generated instance I_1 is *{Weapon:Rifle Type-of-Activity:Open-Attack Occupation-of-Victim:Senator}*. Note that *Weapon:Rifle* and *Occupation-of-Victim:Senator* are terminal values copied from the *S* boundary, and that Open-Attack and Sniping together generalize to Assassination (see Figure 4.5).

Because I_1 matches *H*'s predictor and indeed contains the predictable (or, in this case, something more specific), Ledora expects it to be either positive or negative, but not nonexistent.

If I_1 turns out to be negative, Ledora increases H's confidence, and this experiment is completed. If I_1 turns out to be positive, H's expectation is met but I_1 does not converge Type-of-Activity by more than 50%. In this case, for example, S is generalized by only one level, from Sniping to Assassination (see Figure 4.5). So, Ledora generates a second instance I_2 with the goal of achieving this minimum convergence. I_2 is constructed by copying the values of all attributes except Type-of-Activity from S of the version space. Type-of-Activity in I_2 is given the value Letter-Bomb, a sibling of Assassination. If any value in I_2 is non-terminal, then it is replaced by a terminal descendent of that value. In this example, Letter-Bomb and the values in S, Rifle and Senator, are all terminal values. Hence, the final description of I_2 is *{Weapon:Rifle Type-of-Activity:Letter-Bomb Occupation-of-Victim:Senator}*. This instance should be nonexistent because it pairs *Weapon:Rifle*, which is more specific than *Weapon:Light-Weapon*, with a value other than *Type-of-Activity:Assassination*. If I_2 is indeed nonexistent, Ledora (a) specializes G to Assassination, achieving the complete convergence of the attribute Type-of-Activity, and (b) increases H's confidence. This concludes the experiment. If I_2 turns out to be positive or negative, i.e., *not* nonexistent, H's expectation is violated because, according to the formal definition of H, any instance containing *Weapon:Rifle* and *Type-of-Activity:Letter-Bomb* must be nonexistent. In this case, Ledora discards H and all relations that are more general than H, and this experiment is done. The version space is still updated with this instance, even though the expectation was not met.


Dontcare
G:
Attack-Human
Attack-Things

Intend-Casualty

Midpoint:
Attack-Single
Assassination
Letter-Bomb
S:
Sniping
Open-Attack

Figure 4.5 Partial Generalization Hierarchy for Type-of-Activity.

Let's return to the first instance, I_1, with which Ledora began the experiment. We said Ledora's expectation was that the instance would exist, i.e., be classified as either positive or negative by the world. If I_1 turns out to be nonexistent, the experiment has not worked out as expected, despite the fact that $A{:}a \dashrightarrow B{:}b$ had been verified in theory mode. How, then, could it lead to a "bad" experiment, one in which expectations were not met? In test instance I_1, $\{A{:}a\ B{:}b\ C{:}c\ D{:}d\ ...\}$, some predictive relation other than $A{:}a \dashrightarrow B{:}b$ must be constraining B's value as well. This might be $C{:}c \dashrightarrow B{:}b'$, $D{:}d \dashrightarrow B{:}b''$ (where b' and b'' are siblings of b) or any other attribute that was in the S boundary. Given this uncertain situation, Ledora effectively decides against ever using H again for predicting $B{:}b$. Consequently, Ledora discards H and relations more general than H.

4.6.4.2 Guaranteeing Improvement over Midpoint Method

To ensure that using a verified predictive relation *A:a --> B:b* to create a test instance is better than using the midpoint method, *regardless of the classification of the instance*, the location of the predictable in the version space is constrained in a very specific way. If the predictable is actually the midpoint in the version space, then both methods will generate the same instance. Using the verified predictive relation gains *less* information than using the midpoint method when (a) the predictable *B:b* is more general than the midpoint in the version space, and (b) the instance turns out to be negative for the concept. The reason is because the negative instance will specialize *G* to the predictable's child in the version space. Because the predictable lies above the midpoint, the distance from *G* to the predictable's child will be smaller than (or the same as, if the predictable is the parent of the midpoint) the distance from *G* to the midpoint.

For example, suppose the verified predictive relation is *Weapon:Long-Range --> Type-of-Activity:Attack-Human* and the version space for a simple concept is:

G: {Weapon:Long-Range Type-of-Activity:Attack-Human}
S: {Weapon:Rifle Type-of-Activity:Sniping}

Figure 4.5 shows the generalization hierarchy for the attribute Type-of-Activity. If an instance containing *Type-of-Activity:Attack-Human* turns out to be negative, Type-of-Activity in *G* will be specialized to Intend-Casualty. In this case, the version space is contracted by the distance from Attack-Human to Intend-Casualty. Note that this is shorter than the distance from *G* to the midpoint. Therefore, using the predictive relation to generate an instance actually does worse than the midpoint in the negative case.

Using a verified predictive relation is better than using the midpoint method, regardless of whether the classification is positive or negative, only if the predictable is more specific than the midpoint of the version space. If the instance generated in this case turns out to be negative, the distance contracted will be the distance from *G* to the predictable's child, which is below the midpoint. This distance is greater than half the total distance from *G* to *S*. If the instance turns out to be positive and the predictive relation is valid, then *G* and *S* will converge to the predictable. In this situation, the positive instance generalizes *S* to the predictable. In addition, as mentioned above, Ledora generates a second instance (I_2) containing a sibling of the predictable. According to the definition of a predictive relation, an instance containing the predictor and a *sibling* of the predictable must be nonexistent. An instance that does not exist in the world can be treated by the candidate elimination algorithm as a negative instance of the concept. Therefore, I_2 specializes *G* to the predictable, resulting in the complete convergence of *G* and *S*.

4.7 Summary

Ledora begins in observational mode, monitoring potential predictive relations as it updates its concept definitions. It moves to theory mode whenever proposed predictive relations meet criteria for verification or generalization. Verification tasks in theory mode take precedence over generalization tasks, because the criteria for verifying proposed predictive relations ensure the relations will help learning the concept definitions. Ledora jumps between theory and experimental modes. Whenever a proposed predictive relation is verified, Ledora enters experimental mode immediately, using the predictive relation to advance concept definition learning. When done, it returns to finish any remaining verification or generalization tasks in theory mode. If there are no remaining theory tasks, Ledora returns to observational mode, accepting pre-classified training instances from the world.

Chapter 5

Implementation Details

Ledora is written in INTERLISP-D on a Xerox 1186 Lisp Machine. The top-level control structure that dictates the switching of the three learning modes is implemented as a production system using OPS4 [Forgy, 1979]. This chapter presents the organization of Ledora's predictive relations and the system parameters.

As discussed in Chapter 4, there are two sources of proposed predictive relations: those from the frequency table and those generalized from existing predictive relations. Ledora explicitly records with each proposed predictive relation information about its origin. For predictive relations that originate directly from the frequency table, this information is noted easily when the predictive relation is first recognized in the frequency table. When a predictive relation is formed via generalization, its origin is recorded as the specialized predictive relations that supported the generalization.

In Ledora, predictive relations are arranged in a hierarchy. To do this, we use attribute generalization hierarchies as "skeletons" on which to hang these relations. Specifically, Ledora attaches each predictive relation to nodes in the attribute generalization hierarchies, e.g., the predictive relation *A:a* --> *B:b* is linked with the node *A:a* in the generalization hierarchy for attribute A. This organization of predictive relations makes searching for related predictive relations more efficient for two situations. First, the hierarchical organization helps Ledora to identify quickly the relations that need to be checked for consistency with each incoming instance. Recall that, for an instance to be relevant to a predictive relation, the predictor of the relation must be the same as or more general than the corresponding value in that instance. Under this organization, the relations that need to be checked for consistency are those attached to nodes in the generalization hierarchies that correspond to the attribute-value pairs in the instance and all more general nodes. Second, Ledora can use this organization to identify quickly the predictive relations that are related to disconfirmed relations in order to discard them.[1]

The Ledora system includes many parameters. The following list presents each system parameter, when and how it is used, its value for the experiments we report in the next chapter, and comments about its role or computation.

1. Initial-Frequency

   Purpose: Specifies the initialization value for frequency counts in the frequency table.

   Value: 0.2 (from a scale of 0 to 1).

2. Minimum-Frequency

   Purpose: Specifies the minimum frequency count that a pair of attribute values must attain in the frequency table before Ledora checks them for a potential predictive relation.

   Value: 0.5.

   Comment: This parameter, together with Initial-Frequency, indirectly determine the minimum number of instances that must be seen before Ledora proposes any predictive relation. With the above parameter values, the minimum required number of instances is two. Thus, these two parameters are functionally equivalent to a parameter that says "Wait N instances before checking frequency table for potential predictive relations."

3. Scale-Factor

   Purpose: Determines the maximum size of the frequency increments to values in the frequency table as a result of observing training instances. The size of the increment is limited to be at most a certain proportion, namely the value of Scale-Factor, of the difference between the current frequency count and the upper-bound (1.0).

---

[1] The current implementation of Ledora searches exhaustively the set of predictive relations that Ledora has discovered to date.

Value: 0.3.

Comment: As the frequency count approaches the upper-bound, the actual size of the frequency increments decreases, despite the constant proportion used in computing the increments, because there is less distance to travel. Hence, early instances cause big jumps in the frequency counts, which are followed by smaller increases for later instances. This nature of the function could, to some, be a point of theoretical debate. For example, under a positive accelerating function, initial instances have less impact while subsequent instances have increasing impact until some asymptote is reached.

4. Propagation-Factor

Purpose: Specifies the proportion of frequency increment that is propagated at each successively more general level in the frequency table.

Value: 0.8.

Comment: Values that are more general in the frequency table receive less propagated frequency than values that are more specific. This parameter has a big impact on proposed predictive relations, because it controls how quickly indirect evidence for a predictive relation accumulates.

5. Minimum-Propagate

Purpose: Indicates the minimum amount of frequency increment that Ledora will propagate in the frequency table.

Value: 0.01.

Comment: If the attribute generalization hierarchies are very deep, general nodes in the hierarchies may not receive any propagated frequency, because by the time propagation reaches those nodes, the frequency increment may have dropped below the minimum required for Ledora to propagate it. With more shallow attribute generalization hierarchies, there is a better chance for general nodes in the hierarchies to obtain propagated frequency.

6. Minimum-over-Siblings

   Purpose: Defines, together with the next parameter (Minimum-among-Siblings), when to propose a predictive relation $A:b$ ->$B:b$. Its value specifies the number of times by which the frequency count of $\{A:a\ B:b\}$ must be greater than the average frequency for all $\{A:a\ B:b'\}$ where b and b' are siblings.

   Value: 0.6.

7. Minimum-Among-Siblings

   Purpose: Checks, along with Minimum-over-Siblings, if a predictive relation $A:a$ -> $B:b$ should be proposed. Its value represents the minimum proportion of all $\{A:a\ B:b'\}$ - where b and b' are siblings - that must have a lower frequency count than that of $\{A:a\ B:b\}$.

   Value: 0.8.

8. Minimum-Confidence

   Purpose: Specifies the minimum confidence rating that a proposed predictive relation must reach to qualify as a task in either theory mode or experimental mode.

   Value: 0.6.

   Comment: When any relation passes this confidence threshold, this is the trigger for Ledora to enter theory or experimental mode.

9. Maximum-Missing

Purpose: Determines if Ledora should investigate particular generalizations of predictive relations. Before attempting to generalize predictive relations of the form $A{:}a \rightarrow B{:}b_n$ into $A{:}a \rightarrow B{:}b$ where b is the parent of all b_n's, Ledora verifies that the number of children of $B{:}b$ that are not involved in any of the above relations does not exceed the proportion specified by Maximum-Missing. The case for generalizing relations of the form $A{:}a_n \rightarrow B{:}b$ into $A{:}a \rightarrow B{:}b$ where a is the parent of all a_n's is similar.

Value: 0.2.

10. Minimum-Applicable

Purpose: Specifies the minimum number of concepts to which a proposed predictive relation must be applicable, for Ledora to enter theory mode to verify that relation.

Value: 2.

Comment: It is less likely for relations verified late in a run to satisfy Minimum-Applicable. By that time, most of the concepts are almost completely learned, which means it is less likely that predictive relations will be applicable for reducing the (small) version spaces that remain for those concepts.

It should not be too surprising that so many parameters exist, because abstracting predictive relations is a statistical type of learning. Therefore, Ledora must deal with issues like how many instances should be seen before drawing conclusions, what degree of confidence is needed in a predictive relation before applying it, how that confidence changes upon seeing subsequent instances that confirm or refute the predictive relation, etc. One of the experiments we ran on Ledora examined a simplification of some of these parameters.

Chapter 6

Experiment Results

6.1 Overview

The experiments we ran on Ledora were aimed at answering two major questions. First, we needed to validate Ledora's general design, i.e., to demonstrate that theory mode establishes credible domain relations and that using these verified domain relations in experimental mode creates informative test instances that reduce version spaces of partially-learned concepts. Second, we conjectured that the degree to which the observe-theorize-experiment framework helped learning, relative to an algorithmic approach using the midpoint method, could be a function of the domain structure. So our second goal was to identify and explore some aspects of domain structure that might influence Ledora's learning performance.

Two pilot runs indicated that, unless Ledora encountered instances of all the target concepts in a regular manner and unless the concept definitions had a certain degree of similarity, theorizing and experimenting with predictive relations were rather unsuccessful. Recall that Ledora does not know, a priori, what concepts it must learn or even how many there are; it sets out to learn a concept after observing a positive instance of that concept in the world. The pilot runs, in which concepts appeared randomly, showed that some concepts were not encountered until rather late in the learning experience. This led to an initial bias in the kinds of proposed domain relations that Ledora thought were governing the world. The second pilot run indicated that target concepts needed to have some degree of similarity in their definitions. We also noticed that the target predictive relations that governed the world could either be independent of each other or interact to form additional linear constraints among attributes.

These pilot runs led us to experiment with two independent variables: (a) the order in which Ledora encounters the domain concepts, either randomly or in a fixed, iterative manner, and (b) whether the predictive relations pre-defined to the world simulator (the so-called "target predictive relations" with which all concepts and instances are consistent) are independent of each other or whether they interact to form additional relations among attributes.

We ran a total of 11 experiments. Fixing the total number of learning trials to be 100, Ledora would then "spend" these 100 trials on its 3 modes. We evaluated the degree to which each concept was learned at the end of these 100 trials. Pilot-1 and Pilot-2 are the pilot runs. Two control experiments were run, in which no theorizing or experimenting took place, i.e., Ledora only observed instances that were presented to it by the world. The control experiments provided a baseline against which the benefits of theorizing and experimenting with domain relations were evaluated. Six additional experiments were run to explore the effects of concept presentation order and predictive relation chaining.

The random nature of instance generation in observational mode makes Ledora a non-deterministic system. Each experiment - some manipulation of particular independent variables - consisted of only one run. We recognize that multiple runs are needed to draw definite and statistically reliable conclusions about the effects of the independent variables. For purposes of this thesis. the experiments served (a) to verify that Ledora's design works as intended, (b) to demonstrate that, relative to observing 100 concept instances, the degree of learning is better if some of those 100 trials are spent verifying and using domain relations, and (c) to offer some suggestions about the effects and interactions of concept order and the interdependency among predictive relations.

In the following sections, we first describe the dependent variables in detail. Next, we present the main results of the pilot runs that led us to focus on concept order and predictive relation chaining. We then describe and discuss the results found with the experiments that manipulated these factors. Finally, we look at simplifying some system parameters in these experiments and the effects this has.

6.2 Dependent Variables

As noted above, we held each experimental run to 100 trials. This means that, regardless of whether or not Ledora learned all the concepts completely, the run was terminated after instance 100. Thus, one important dependent variable was how well-learned each concept was at the end of the run. This is the key benchmark measure (Figure 6.5c) by which we can evaluate Ledora's general design and its performance in specific experiments.

A concept's learnedness was determined by the distance remaining between *G* and *S* in its version space. The smaller the distance between *G* and *S*, the more well-learned the concept. We use the notation of a vector of distances to summarize the status of a version space, for instance, (0 1 0 2). Each number in the vector represents the distance between *G* and *S* of a specific attribute of the concept. Suppose the numbers in (0 1 0 2) correspond to the attributes A, B, C, and D respectively. The meaning of (0 1 0 2) is that attributes A and C of the concept are fully converged (learned) on a single value while the distances between *G* and *S* of attributes B and D are one and two, respectively. To measure the overall learnedness of all concepts, we compute a metric called the *concept divergence index* by totalling the distance vectors of all domain concepts. The lower the divergence index, the more well-learned overall the entire concept set.

Besides concept learnedness, we tracked a number of other measures as descriptive variables; i.e., they clarify what went on during the experiments. One set of descriptive dependent variables characterized the predictive relations that Ledora had proposed. We counted the total number of proposed predictive relations and then separated this group into those relations that were equal to or supported the target predictive relations, and all other relations, which we called "spurious" relations. A predictive relation H_1 *supports* another predictive relation H_2 if H_1's predictor and predictable are either the same as or more specific than their counterpart in H_2. For example, *Occupation-of-Victim:Politician --> Type-of-Activity:Assassination* is a supporting relation for *Occupation-of-Victim:Politician --> Type-of-Activity:Attack-Single*. Note that the predictor is identical in both predictive relations and that Assassination is more specific than Attack-Single. It is important to credit Ledora for discovering supporting predictive relations, because these are often the first steps to discovering the final target relations. With continuous exposure to instances that are consistent with supporting predictive relations, a target predictive relation will eventually emerge, via either the propagation of frequency counts or generalization steps.

Spurious predictive relations were not necessarily incorrect or useless in experimentation. We concluded, somewhat unexpectedly, that the interactions among the concept definitions themselves may cause other valid predictive relations to emerge in addition to those that we pre-defined for the world simulator. For each type of proposed predictive relation, we measured the average and median confidence.

Another major set of descriptive dependent variables evaluated the quantity and quality of the work done in each of the three modes. The first variable was the percentage of the 100 trials spent in observational, theory, and experimental modes. We then broke the trials spent in theory mode down into trials spent on testing proposed predictive relation credibility and trials spent on generalizing non-exclusive sibling relations, i.e., relations whose predictors (or predictables) constitute some but not all the immediate specializations of some attribute-value pair. For the trials spent on credibility tests, we measured how many of such tests were successful. Recall that, in theory mode, Ledora takes a proposed predictive relation and generates an instance that it expects is nonexistent, if the predictive relation is true. If this expectation is met, then this is a successful test. Indirectly, this success count measured how many proposed predictive relations that Ledora theorized about turned out to be "true" relations. The other qualitative measure of theory mode work was the percentage of generalizations it formed that were equal to or supported the target predictive relations. We also measured the percentage of successful experiments in experimental mode. Recall that, in experimental mode, Ledora uses a verified predictive relation to generate a sequence of instances aimed at reducing the version space of partially-learned concepts. It has certain expectations (as described in Chapter 4) that the instances it generates are either positive, negative, or nonexistent. Each time one of these expectations was met counted as a success. This indirectly measured the quality of the predictive relations that were verified in theory mode.

6.3 Pilot Experiments

6.3.1 Pilot-1

Pilot-1 was the first run made with Ledora. Concepts were introduced randomly in observational mode. Figure 6.1 shows the target concept definitions and the target predictive relations used for Pilot-1, and Figure 6.2 tabulates some of Pilot-1's results. The most important result is the low degree of success in using verified predictive relations in experimental mode. Specifically, the classification of the instances generated in this mode met the expectation of the corresponding predictive relations only 43% of the time. This implied that Ledora ran primarily "bad" experiments because the predictive relations that were previously tested in theory mode were actually invalid.

Concept Names	Concept Definitions				
	Weapon	Crime-Location	Occupation-of-Victim	Type-of-Activity	Crime-Site
Religious-Terror	Long-Range	Caribbean	Govnt-Employee	Attack-Single	Public-Place
High-Profile	-	Mexico	Politician	Attack-Single	Urban
Successful-Demand	Light-Weight	America	Govnt-Employee	Attack-Things	-
Political-Terror	Grenade	Europe	Govnt-Employee	Theft	-
Urban-Terror	Gun	C-America	Politician	Attack-Single	Public-Place
Overt-Terror	Short-Range	Eastern-Bloc	Law-Official	Bomb-Attack	Rural
Govnt-Support	-	-	Industry	Bomb-Attack	Entertain

a) Target Concept Definitions.

Crime-Site:Tourist-Point --> Type-of-Activity:Attack-Single
Type-of-Activity:Attack-Single --> Weapon:Light-Weight
Weapon:Submachine-Gun --> Crime-Site:Public-Place
Crime-Site:Commercial --> Occupation-of-Victim:Govnt-Employee
Occupation-of-Victim:Politician --> Weapon:Light-Weight
Crime-Location:C-America --> Crime-Site:Urban
Crime-Location:Eastern-Bloc --> Crime-Site:Rural
Crime-Site:Rural --> Type-of-Activity:Bomb-Attack
Crime-Location:America --> Weapon:Light-Weight
Occupation-of-Victim:Judge --> Weapon:Light-Weight
Crime-Location:America --> Crime-Site:Urban

b) Target Predictive Relations

Figure 6.1 Concepts and Predictive Relations used in Pilot Experiments.

Experiment Names	% Distribution of Instances			% Distribution of Work in Theory Mode		Success Rate in Theory Testing (%)	% of "good" Sibling Generalizations		Success Rate in Experim Mode(%)
	Theory	Experim	Observ	Generalizing	Testing		Exclusive	Non-excl.	
P-1	48	13	39	62	38	28	8	0	43
P-2	35	7	58	60	40	21	16	0	50

Figure 6.2 Results of Experiments Pilot-1 and Pilot-2.

We noticed from Pilot-1's trace that Ledora did not encounter some of the domain concepts until late in the run. This is due to the random order for choosing concepts for presentation in observational mode. As a result, the predictive relations proposed earlier in the run were based on observing instances of only a subset of the domain concepts. In other words, these proposed predictive relations were not representative of the whole domain. These biased predictive relations would pass the credibility test during theory mode (given a biased world of limited concepts). However, these relations proved faulty when Ledora used them to experiment on new concepts that emerged later in the run. We suspected that this was part of the reason for the poor performance of verified predictive relations in experimental mode.

Consequently, we decided to explore the effect of concept order on learning performance. We used two methods for determining the order in which Ledora would see the target concepts in observational mode: the *iterative concept sampling method* and the *random concept sampling method.* The former method selects a concept for instance presentation by iterating through the list of target concepts in a fixed order. For example, if there are three target concepts, then the order for showing instances of these concepts might be 1-2-3-1-2-3-1-2-3-.... The random method selects concepts at random during observational mode, corresponding to the approach taken in Pilot-1.

Investigating this order difference has some interesting real-world correspondences. Random concept sampling can be viewed as analogous to a discovery learning task in which the learner has no prior knowledge of which concepts he is supposed to learn. The learner simply encounters new events of different types in a random fashion. He must rely on the "world" for input, with no control over what that input will be. In contrast, we can view iterative concept sampling as a controlled experiment that is performed in a laboratory. In this case, the learner or experimenter knows precisely what concepts he is supposed to learn, and can investigate them in some orderly fashion of his choosing. The second pilot run, Pilot-2, used iterative concept sampling.

6.3.2 Pilot-2

In experiment Pilot-2, both the target predictive relations and the target concepts were the same as those in Pilot-1(see Figure 6.1). The major design difference between the two pilot experiments was that concepts were presented iteratively in Pilot-2 as opposed to randomly in Pilot-1.

Figure 6.2 above summarizes results about theory and experimental mode activities for both experiments. Contrary to expectation, the success rate in experimental mode did not improve under iterative concept presentation. We then noticed that the verified predictive relations with which Ledora experimented were neither the target predictive relations nor relations supporting the target ones. This is because, although the world was governed by the target predictive relations, Ledora simply did not recognize these relations due to their low confidence ratings.

After a close examination of the target concept definitions, we discovered that the degree of overlap among these concepts was relatively small. By concept overlap, we mean the degree of similarity among the target concept definitions as measured by two features: the number of common attributes between two definitions and the relationship between the corresponding values of common attributes. Concepts whose values of common attributes are the same or more general than each other are defined as more similar than concepts whose values of common attributes are mutually exclusive. For example, suppose the target definition of the concepts C_1, C_2 and C_3 are respectively

{Occupation-of-Victim:Politician Type-of-Activity:Attack-Human},
{Occupation-of-Victim:Politician Type-of-Activity:Letter-Bomb}, and
{Occupation-of-Victim:Industrialist Type-of-Activity:Attack-Things}.

C_1 and C_2 have a high degree of overlap, since Politician is more general than President, and Attack-Human is more general than Letter-Bomb. C_1 and C_3 have a relatively lower degree of overlap because Industrialist and Politician, and also Attack-Things and Attack-Human, are mutually exclusive values of the Occupation-of-Victim and the Type-of-Activity attributes, respectively.

We conjectured that a low degree of concept overlap was responsible for Ledora's failure to abstract the target predictive relations that governed the world. The rationale behind this is as follows. Because the concept definitions were very different from each other, a target predictive relation was often supported by only a small subset (possibly just one) of the concepts. This caused the target predictive relations abstracted by Ledora to have a relatively low degree of confidence. For example, suppose the concepts to be learned include groceries, furniture, and animals. Then a target predictive relation such as "An object is more expensive if it is canned than if it is fresh" is only relevant to one category of concepts, namely, groceries. This conjecture was supported by the fact that, although Ledora eventually discovered some of the target predictive relations (or target-supporting predictive relations), it did so rather late in the run. By that time, the domain concepts were so well-learned that the abstracted predictive relations had no useful impact on learning. For example, if Ledora has just discovered the predictive relation *Occupation-of-Victim:Politician --> Weapon:Gun*, and if it already knows the value of Weapon in all the concepts involving this attribute, this piece of knowledge comes too late to be useful. The delay in noticing the target predictive relations and their low confidence prevented them from being explored in theory mode and applied in experimental mode. This interpretation is also consistent with the observation that most of the predictive relations applied in experimental mode were spurious relations.

In the light of these results, we modified the target concept definitions so that they would overlap to a larger extent. In hindsight, this made perfect sense: in a situation of learning multiple concepts simultaneously, relations proposed from observing instances of one concept have little use if that concept has no similarity to the rest of the concepts. From the perspective of evaluating our general design, it also seemed reasonable to ensure that the world was stacked in Ledora's favor: if it could not benefit from domain relations in the most favorable situation (highly similar concepts) then the basic ideas would be suspect. For the remainder of the experiments, then, all the concept definitions were redesigned to have a larger degree of overlap. These new concept definitions are given in Figure 6.3 below.

Concept Names	Concept Definitions				
	Weapon	Crime-Location	Occupation-of-Victim	Type-of-Activity	Crime-Site
Religious-Terror	Long-Range	Caribbean	Politician	Attack-Single	Commercial
High-Profile	-	Mexico	Law-Official	Attack-Single	Travel-Center
Successful-Demand	Long-Range	America	Govnt-Employee	Attack-Things	-
Political-Terror	Long-Range	Latin-America	Govnt-Employee	Attack-Multiple	-
Urban-Terror	Long-Range	C-America	Politician	Attack-Single	Commercial
Overt-Terror	Long-Range	Mexico	Politician	Assassination	Commercial
Govnt-Support	Long-Range	C-America	Politician	-	Commercial

Figure 6.3 Revised Target Concept Definitions.

After studying the target predictive relations defined for Pilot-2 (shown in Figure 6.1), we also noticed that chaining existed among some target predictive relations. The term *chaining* refers to situations where the predictable of one predictive relation is either the same as or more specific than the predictor of another predictive relation. An example of the first case is *Crime-Site:Tourist-Point --> Type-of-Activity:Attack-Single* with *Type-of-Activity:Attack-Single --> Weapon:Light-Weight*. An example of the second case is *Type-of-Activity:Attack-Single --> Weapon:Submachine-Gun* with *Weapon:Light-Weight --> Crime-Site:Public-Place*, where Submachine-Gun is a more specific value of Light-Weight. The notation we use for chaining is *A:a --> B:b --> C:c*. As more predictive relations are involved, the "chain" may grow longer, for example, *A:a --> B:b --> C:c --> D:d --> ... --> Z:z*.

Uncertain of its effect on learning, we decided to explore further the issue of chaining in later experiments.

The next section presents experiments designed to investigate how iterative and random sampling of concepts impact learning. The section after that discusses how predictive relation chaining affects learning.

6.4 Effects of Presentation Order

This section describes the effects that iterative and random concept sampling have on learning. The first set of experiments consists of one iterative-chaining (I-C) run and two random-chaining runs (R-C and R-C'). The reason for using two experiments to investigate random concept sampling is to offset some of the variability inherent in that method. The second set of experiments includes one iterative-no-chaining (I-NC) and two random-no-chaining runs (R-NC and R-NC'). For each experiment in this set, the target predictive relations did not yield any chaining relationships among each other.

Figure 6.4 shows the target predictive relations used in the chaining experiments and those used in the no-chaining experiments. The target concept definitions were the same in all six experiments (see Figure 6.3). Despite the common concept definitions, one must be careful in collapsing across the chaining/no-chaining dimension, because the target predictive relations were different in each case. In fact, some results we discuss below suggested an interaction between presentation order and predictive relation chaining.

Crime-Site:Public-Place --> Type-of-Activity:Attack-Single
Type-of-Activity:Attack-Single --> Weapon:Long-Range
Weapon:Long-Range --> Crime-Site:Commercial
Crime-Site:Commercial --> Occupation-of-Victim:Politician
Occupation-of-Victim:Govnt-Employee --> Weapon:Long-Range
Crime-Location:C-America --> Crime-Site:Public-Place
Occupation-of-Victim:Law-Official --> Weapon:Long-Range
Crime-Location:America --> Weapon:Long-Range

a) Target Predictive Relations in Chaining Experiments.

Crime-Site:Public-Place --> Type-of-Activity:Attack-Single
Occupation-of-Victim:Govnt-Employee --> Crime-Location:America
Occupation-of-Victim:Law-Official --> Weapon:Long-Range
Crime-Site:Commercial --> Occupation-of-Victim:Politician
Crime-Location:C-America --> Weapon:Long-Range
Crime-Site:Commercial --> Type-of-Activity:Attack-Single
Crime-Site:Public-Place --> Crime-Location:America

b) Target Predictive Relations in No-Chaining Experiments.

Figure 6.4 Predictive Relations in Chaining & No-Chaining Experiments.

Figure 6.5 tabulates the results of the experiments mentioned above. For each of the two pairs of random experiments, R-C/R-C$'$ and R-NC/R-NC$'$, statistics on the learning modes and predictive relations are the means of the two experiments in each pair. Version space statistics are reported separately for each experiment.

Experiment Names	% Distribution of Instances			% Distribution of Work in Theory Mode		Success Rate(%) in Theory Testing	% of good Sibling Generalizations		Success Rate in Experim Mode(%)
	Theory	Experim	Observ	Generalizing	Testing		Exclusive	Non-excl	
I-C	33	24	43	55	45	33	17	33	92
R-C	48	22	31	56	44	27	18	28	87
I-NC	52	19	29	63	37	21	20	27	90
R-NC	39	19	42	38	63	21	18	38	85

a) Mode Results.

Experiment Names	Total # of Relations	Target Relations			Transitive Relations			Spurious Relations		
		% of Total	Ave Confid	Median	% of Total	Ave Confid	Median	% of Total	AveConfid	Median
I-C	424	19	.67	.74	18	.49	.47	63	.56	.59
R-C	452	16	.66	.73	20	.59	.60	64	.59	.65
I-NC	311	16	.57	.57	-	-	-	84	.60	.62
R-NC	428	19	.60	.64	-	-	-	81	.59	.63

b) Predictive Relation Results.

Experiment Names	Concept Names							Divergence Index
	Religious Terror	Urban Terror	High Profile	Success Demand	Political Terror	Overt Terror	Govnt Support	
I-C	(0 0 0 0 1)	(0 0 0 0 2)	(0 5 0 0)	(0 3 0 0)	(0 0 0 2)	(0 0 3 0 0)	(0 0 0 0)	16
R-C	(4 4 0 3 0)	(0 0 0 1 0)	(1 1 0 0)	(0 1 1 0)	(0 0 0 -1)	(2 1 2 0 0)	(0 0 0 0)	22
R-C'	(0 0 1 0 0)	(1 0 0 0 0)	(1 0 1 0)	{(1 3 1 1) (3 3 0 1) }	(0 0 2 0)	(0 0 4 2 0)	(0 1 0 1)	27
I-NC	(1 1 0 0 0)	(0 0 0 0 0)	(2 5 1 1)	{(0 1 3 0) (1 0 3 0) }	(1 3 0 0)	(0 1 0 1 0)	(2 0 0 0)	27

R-NC	(0 1 0 0 0)	(0 0 0 0 0)	(2 1 0 0)	(1 0 0 1)	(1 0 1 4)	(0 0 0 0 0)	{
							(2 0 0 0)
							(1 0 0 2)
							(5 0 0 1)
							}
								26

R-NC'	(0 0 0 0 0)	(1 0 0 1 0)	(0 2 3 2)	(1 1 0 2)	(0 0 1 1)	(1 0 1 2 1)	(0 0 0 0)	20

c) Version Space Results.

Figure 6.5 Results for Iterative versus Random Concept Presentation.

The most important result to notice from Figure 6.5a is that the success rate of Ledora's experiments was, at last, quite high. This means that experiments aimed at generating instances for concept learning were based on valid predictive relations. As discussed below, not all the predictive relations used for experiments were necessarily our target relations.

The results on version space convergence (Figure 6.5c) showed that concepts were somewhat better learned with the iterative method than with the random method. The one exception to this conclusion was that the concept Successful-Demand was poorly learned in experiment I-NC. That result, however, was due to one negative instance, appearing late in the run, that caused the *G* boundary set to split. This dependent measure is particularly sensitive to the non-deterministic nature of instance creation during observational mode. In general, we would expect better concept learning under iterative presentation because, as explained below, credible predictive relations were proposed and exploited sooner under that method. However, multiple experimental runs under each condition are needed to confirm this hypothesis.

Now that we've examined how iterative versus random concept presentation affects concept learning, we next study some descriptive data that will give us a better picture of what happened during the experiments. First, we consider the data about the proposed predictive relations (Figure 6.5b). For the chaining experiments, we defined a new category of predictive relations: transitive predictive relations. If the target predictive relations are $A \text{-->} B$ and $B \text{-->} C$, then a transitive predictive relation would be $A \text{-->} C$. Ledora proposed more predictive relations of all three categories (target, transitive and spurious) with random concept presentation than with iterative concept presentation. Specifically, the number of target/transitive/spurious relations for experiments R-C and I-C are 82/89/326 and 79/77/268 respectively. Moreover, the number of target/spurious relations for experiments R-NC and I-NC are 87/329 and

51/260 respectively. The reason Ledora discovered more target relations and transitive relations under random concept presentation is this: at various times in the run, the world by chance provided Ledora with instances of only *some* concepts that supported *some* particular target predictive relations. This constituted a bias toward recognizing those target predictive relations as well as transitive relations (if there is chaining among the target relations). Ledora was also more likely, under the random method, to discover spurious predictive relations early in the run when only a subset of the domain concepts is seen. The reason is that concepts that might otherwise counteract these spurious relations were, by chance, not encountered until later.

Despite the increase of proposed predictive relations in all three categories, the percentages of proposed relations in each category over the total number of proposed relations remain about the same for both random presentation and iterative presentation. Specifically, the percentage of target/transitive/spurious relations in R-C is 16/20/64 as compared to 19/18/63 in I-C. In addition, the percentage of target/spurious relations in R-NC is 19/81 as compared to 16/84 in I-NC. This indicates that a quantitative increase does not necessarily change the proportion of proposed predictive relations in each category.

Given the interpretation that random encounters with concepts bias Ledora to propose spurious predictive relations, it seemed curious that Ledora did not propose *more* spurious relations, relative to target and transitive relations, with random presentation than with iterative presentation. However, the timing of when Ledora proposes which category of predictive relations does support the notion of an initial bias towards spurious domain relations. By trial 25, Ledora had proposed about 100 predictive relations under both presentation methods. Under iterative presentation method, 10 of the 100 predictive relations were target or transitive relations. Under random presentation method, only 6 of the 100 predictive relations were target or transitive relations. This makes sense because if concepts are presented to Ledora randomly, the discovery of target and transitive relations are necessarily stretched out over the entire 100-trial run.

Recall that Ledora proposed more predictive relations overall under random presentation method than under iterative presentation method. Yet, we note above that, by trial 25, Ledora had abstracted about the same total number of proposed predictive relations with both iterative and random presentations. A valid question to ask is when Ledora discovered all the extra relations with random presentation. Figure 6.6 shows, for experiments I-C and R-C/R-C', the number of relations proposed in each category at different time intervals of the experiment. Note that the increase of target and transitive relations (and relations supporting them) happened in the second half of the experiment (between trials 51 and 100), and that the increase of spurious relations happened in the first and last 25 trials. We have already explained why there are more spurious relations early in the experiment. Below, we try to account for the increase of relations as the experiments went along.

Under the iterative presentation method, the regular presentation of instances of each concept counteracted the bias to form spurious predictive relations. This is especially true during later stages of the learning experience when Ledora had seen many instances of each concept. Most of the target and transitive relations were abstracted early in the run because iterative presentation allows Ledora to see all the concepts in the earliest possible moment. These two facts explain why the number of relations are less under iterative presentation during the later stages. Under random presentation, we conjecture that each new concept introduced to Ledora "triggered" the discovery of a whole new set of target, transitive, and spurious relations. In other words, we can view a random experiment as a series of independent learning experiences demarcated by which concepts it has seen so far. For example, one learning experience can be that of seeing only 50% of the concepts; another can be that of seeing the last 50%. To verify the above conjecture, we examined when the different concepts were introduced during the random presentations, and related this information to the number of relations proposed around that time. We found that, on average, 64% of all concepts were introduced between trials 1 and 25, 22% between trials 26 and 50, and the remaining 14% between trials 51 and 75. The late introduction of some concepts coincides with the surges in proposing both target and spurious relations in the last half of the run.

	Instance Interval			
Relation Category	1 - 25	26 - 50	51 - 75	76 - 100
Target(-Supporting) + Transitive(-Supporting)	51	51	36	18
Spurious Relations	45	64	100	59

a) Experiment I-C.

	Instance Interval			
Relation Category	1 - 25	26 - 50	51 - 75	76 - 100
Target(-Supporting) + Transitive(-Supporting)	43	32	43	45
Spurious Relations	54	47	91	98

b) Averages for Experiments R-C and R-C$'$.

Figure 6.6 Timetable for the Discovery of Relations in I-C and R-C/R-C$'$.

Figure 6.5b also reveals that the spurious predictive relations discovered in all the experiments had a relatively high confidence compared to the target relations. For instance, in experiment I-NC, the average confidence ratings of spurious relations and target relations are .60 and .57 respectively. This is a little disturbing because one normally expects a lower confidence for the spurious predictive relations. However, there are two things to note. First, a spurious relation is, by definition of our categories, any predictive relation not related to the targets pre-defined to the world simulator. As mentioned earlier, some of these spurious predictive relations led to perfectly accurate experiments throughout the run. This suggests that there were some unplanned domain relations due to interactions among attributes in concept definitions and attributes in the target predictive relations. It is difficult to validate this conjecture, but it seems reasonable to expect some valid constraining relations to evolve "naturally," especially because the concept definitions were very similar. Second, about one-third of all spurious relations were very general (the relation's predictor and/or predictable correspond to an immediate child of the root node). These general relations often accumulate a high confidence, say greater than 0.7, because they can be supported by many instances. For example, the proposed predictive relation "If a terrorist event happens on land, then it usually involves an attack on human beings" may be supported by many terrorist events. It is also worth mentioning that although these relations have very high confidence ratings, they are seldom used to generate experiment instances. Recall that for a predictive relation to be applicable to a concept, the predictable must lie between *G* and *S* inclusive. Quite often, a general relation is inapplicable to all the concepts because its predictable is more general than the corresponding value in each concept's *G* boundary.

All the experiments followed a common execution pattern that can be characterized by three stages. In stage 1, Ledora remains entirely in observational mode. Because it does not know any predictive relations at the start, there is no choice but to passively observe instances provided by the world. As more instances appear, Ledora begins to propose predictive relations. Stage 2 begins when some proposed predictive relation gains enough confidence to be applied for learning. This stage is marked by repeated entries into theory mode to test the credibility of proposed predictive relations and into experimental mode to generate instances based on credible relations. More specifically, stage 2 iterates over one theory mode instance, created to test a proposed predictive relation's credibility, followed by one or more experimental mode instances, generated using the relation, to reduce some concept's version space. Stage 3 begins when Ledora no longer enters experimental mode. In this final stage, Ledora reverts to watching instances in observational mode. Occasionally, Ledora engages in theory mode to generalize predictive relations.

There are several reasons for this pattern of activities. First, most target-supporting predictive relations have already been applied in stage 2 to reduce the version spaces. Predictive relations proposed in stage 3 are mostly spurious relations and therefore do not pass the credibility test in theory mode that is the prerequisite for applying them in experimental mode. Second, the domain concepts are often well-learned at this late stage (i.e., the distance between *G* and *S* is quite small). Hence, even if a credible predictive relation emerges in stage 3, the version spaces may be already converged to an extent that the relation has no role to play. Recall that Ledora is constrained to apply verified predictive relations to generate test instances as experiments only if doing so would yield a version space improvement *better* than midpoint.

There are some interesting deviations within this general pattern of stages as a function of iterative versus random sampling. Figure 6.7 presents a graph of when Ledora entered each mode as a function of trial in the I-NC experiment. Figure 6.8a depicts a similar graph for one of the R-NC experiments. Figure 6.8b shows when, in the R-NC experiment, Ledora encountered instances of each concept. Note the different patterns between trials 30 and 80: stage 2 extends over more trials under random presentation (from trial 7 to 62) relative to iterative presentation (from trial 8 to 37). This trend holds for chaining experiments as well. The reason for the increase is that some concepts, and hence some credible predictive relations, may not appear until late in the experiment. As long as credible relations are still emerging, Ledora continues to enter experimental mode. For example, Ledora does not encounter the concept Religious-Terror until trial 33 (see Figure 6.8b). There is a corresponding switch back into theory and experimental modes when these concepts are encountered. On the other hand, experiments using the iterative method usually exhaust early the detection of credible predictive relations.

Predictive relations that led to successful experiments in experimental mode belonged to all three types: target relations, transitive relations, and spurious relations. The fact that some spurious relations were actually successful shows Ledora's ability to abstract predictive relations unforeseen in the domain "setup" and yet beneficial to concept learning. On the other hand, spurious relations may also fail in experimental mode. In fact, the unsuccessful experiments in experimental mode involved exclusively the spurious predictive relations. This is sensible because predictive relations of the other two categories either directly or indirectly supported the target relations and hence are less likely to fail in their prediction.

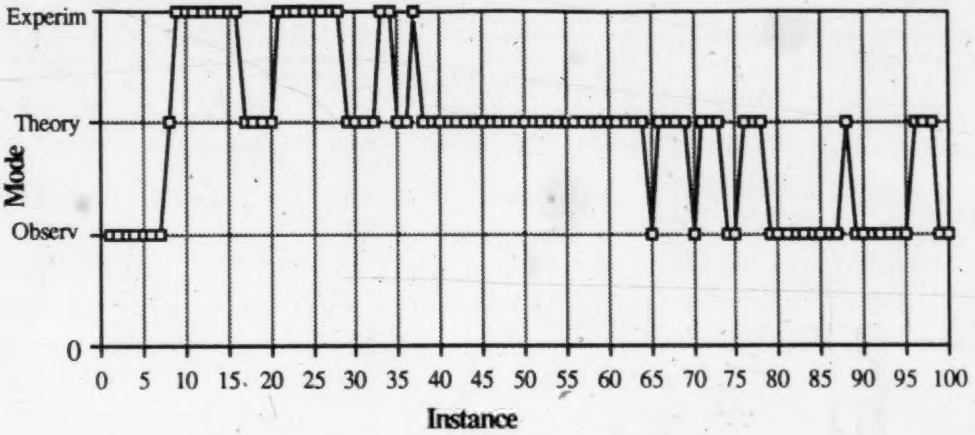


Figure 6.7 Mode Activities in Experiment I-NC.

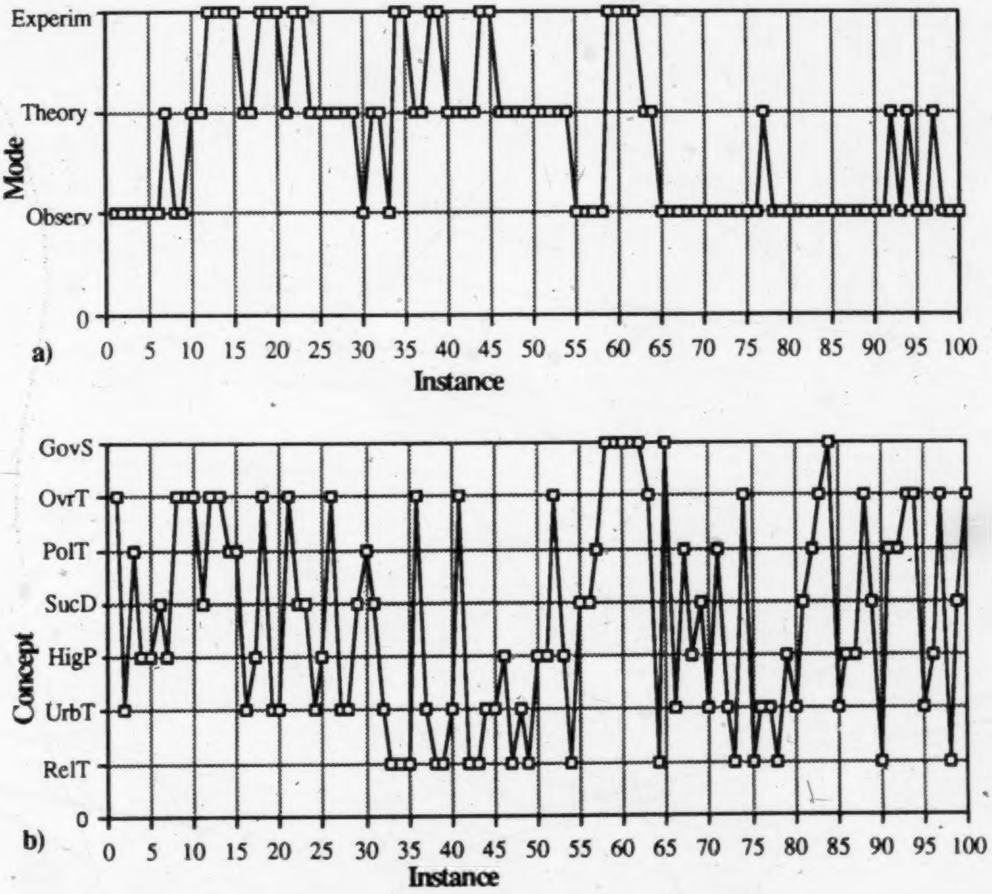


Figure 6.8 Mode Activities and Concept Order in Experiment R-NC.

6.5 Chaining Effects and Interactions with Presentation Order

We now re-examine the data in Figure 6.5 to consider the effects of chaining among target predictive relations versus no chaining. Specifically, we compare the results of I-C and R-C/R-C$'$ - experiments with chaining - with the results of I-NC and R-NC/R-NC$'$ - experiments without chaining - respectively. The total number of predictive relations proposed in experiments with chaining (424 and 452 for I-C and R-C/R-C$'$ respectively) was greater than the number proposed in experiments without chaining (311 and 428 for I-NC and R-NC/R-NC$'$ respectively). This increase in the number of proposed predictive relations was largely due to discovering transitive predictive relations and those that support such relations. For example, *Weapon:Long-Range --> Crime-Site:Commercial* and *Crime-Site:Commercial --> Occupation-of-Victim:Politician* are two target predictive relations in the R-C experiments. During those experiments, Ledora discovered the transitive relation *Weapon:Long-Range --> Occupation-of-Victim:Politician.*

There was an interesting pattern of results in the distribution of instances between theory and observational modes as a function of chaining under iterative presentation. More trials were spent in theory mode in the I-NC experiment than in the I-C experiment (52 trials vs. 33 trials). Figures 6.7 and 6.9 show mode activities as a function of trial for experiments I-NC and I-C respectively. Note that in the I-NC experiment, there was continuous theory work between trials 35 and 65. On the contrary, in the I-C experiment, the theory work came after trial 57. The extra theory trials in experiment I-NC are taken away from trials that would otherwise be spent in observational mode. This extra theory work was spent on generalizing predictive relations. There were nearly twice as many generalization attempts with I-NC (33) than with I-C (18), despite the fact that the total number of proposed predictive relations was greater under the chaining condition. Because Ledora discovered the same percentage of credible (non-spurious) predictive relations in both situations, the extra generalizations in I-NC must be based on spurious relations. This increase in spurious relations also shows up in the type of verified predictive relations Ledora used to generate test instances in experimental mode. Collapsing across presentation order, the percentage of spurious relations used for experiments in the chaining situation was 19% (3 out of 16) versus 57% (8 out of 14) in the no-chaining situation.

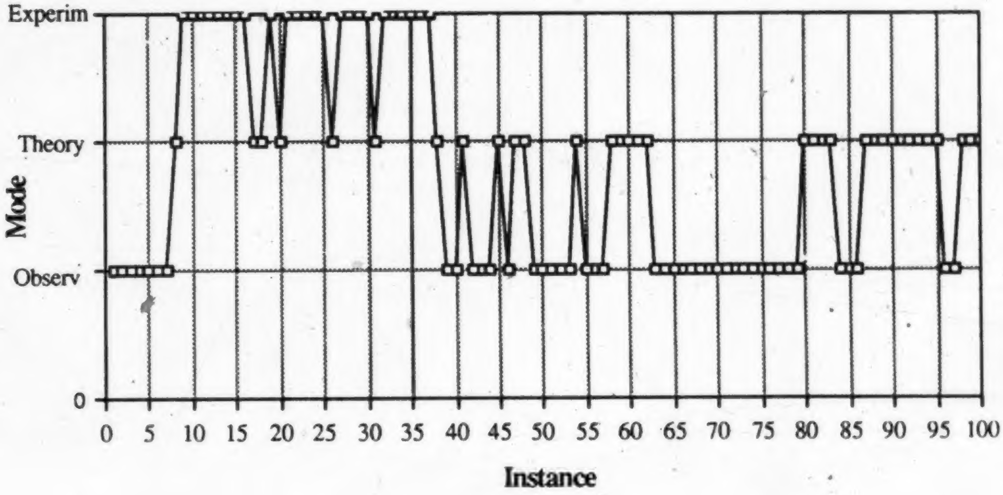


Figure 6.9 Mode Activities in Experiment I-C.

One conjecture about why there were more spurious generalizations with I-NC is the following. Recall that the concept definitions were the same in both chaining and no-chaining experiments. These concept definitions as a whole support the predictive relations that hold true in the domain. Suppose the total number of valid predictive relations supported by these concept definitions is M. However, not all M relations may be explicitly specified to the world simulator as target predictive relations. Some of these relations might emerge as a function of interacting concept definitions.

Suppose that, among the M valid relations, (a) N are target predictive relations, and (b) P are transitive relations (P can be zero). The difference (M - N - P) is the number of valid relations that Ledora would classify as spurious. In no-chaining situations, P is zero; in chaining situations, P is greater than zero. Therefore, more predictive relations would be classified as spurious in no-chaining situations than in chaining situations. The extra spurious relations caused more generalization attempts based on those spurious relations.

If the above conjecture is true, then we should find that most of the spurious relations are actually valid relations in the world. Indeed, we found that the instances generated by these spurious predictive relations led to successful experiments 65% of the time.

In no-chaining situations, there were more spurious relations and hence more generalization attempts based on these relations. However, we still need to explain why there were *more* generalization attempts, regardless of the category of relations they belong to. Suppose $B{:}b_1$, $B{:}b_2$, and $B{:}b_3$ are the immediate specializations of $B{:}b$. In chaining situations, the pair $A{:}a$ and $B{:}b$ can gain co-occurrence frequency counts via interacting relations. Specifically, instances that support $A{:}a$ --> $C{:}c$ and $C{:}c$ --> $B{:}b$ also support $A{:}a$ --> $B{:}b$. On the contrary, the target predictive relations in no-chaining situations are independent and non-interacting. Therefore, fewer predictive relations contribute to the frequency count of the potential relation $A{:}a$ --> $B{:}b$. This lessens the chance of noticing $A{:}a$ --> $B{:}b$ from the frequency table. Consequently, Ledora is left with only the specialized relations: $A{:}a$ --> $B{:}b_1$, $A{:}a$ --> $B{:}b_2$, and $A{:}a$ --> $B{:}b_3$. However, from these relations, Ledora does a generalization step to obtain the relation $A{:}a$ --> $B{:}b$. Hence, more general predictive relations were proposed by generalization instead of through the frequency table. This explains why there were more generalization attempts in no-chaining situations.

6.6 Summary

Random presentation caused relatively more spurious predictive relations to emerge during the early trials when Ledora experienced a bias with respect to what kinds of concepts exist in the world. Theory mode effectively filtered these out with credibility testing, so there is little impact on experimental work. However, these spurious predictive relations, by their very existence, required this extra theory effort.

In the I-NC experiment, because the target predictive relations were independent and non-interacting, fewer relations contributed to the frequency counts of potential relations that involved general attribute-value pairs. Yet, the presence of predictive relations involving immediate specializations of those attribute-value pairs led Ledora to more generalization attempts. Although the extra predictive relations proposed by generalization were mostly spurious, they were credible a good deal of the time, and in fact, did yield successful test instances in experimental mode. Determining whether this finding is stable, whether it is a function of chaining *per se*, or whether it is due to the particular target predictive relations and concept definitions used, would require many more experimental runs with the same and different target predictive relations.

Finally, there is some suggestion that concept learning was slightly better under iterative presentation. This is an intuitive result, but because the size of the *G* boundary is so sensitive to the occurrence of a negative instance, this conclusion also needs the support of more experimental runs to be definitive.

6.7 Control Experiments

The above results demonstrate that the verified predictive relations led to highly-successful experiments on test instances. Specifically, the classification of the instances in experimental mode matched the expectation of the verified predictive relations that had generated those instances. The matched expectations indicate that, in general, the relations proposed and verified by Ledora were useful and valid. A reasonable question is whether the benefit of using a domain model - a set of abstracted predictive relations - is worth the cost of formulating it.

"Cost" is a difficult measure to define. Clearly, there is additional storage and processing cost in terms of maintaining all the frequency information that allows Ledora to detect domain relations. However, there is a pragmatic measure of cost that is more relevant to a learning task: the number of training instances required to learn the concepts. If, by abstracting and using a domain model, Ledora can learn multiple concepts from fewer training instances, then the extra effort for maintaining frequency data is justified. This is particularly valid for learning situations in which a learner has no control over how many or how often concept instances are encountered (e.g., trying to learn different types of enemy submarines from radar information, and waiting for the occasional example to appear). It is also true for learning situations in which the learner can generate test instances (run experiments), yet there is an associated "cost" of either generating the instance or getting the classification (e.g, there is a real time or a real money cost for either formulating the question or getting the answer). Thus, for a learning-by-example system, the number of training instances is perhaps the most critical measure of the cost and benefits associated with a particular framework.

In considering the number of training instances Ledora uses, it is important to recall that Ledora explicitly sets aside its goal of learning domain concepts when it enters theory mode. In theory mode, it "uses up" training instances for the purpose of verifying and generalizing proposed domain relations. The instances it generates in theory mode do affect the version spaces, but they are created very conservatively by copying attribute-value pairs from the *S* boundary. Only one attribute is then changed. This means that a theory mode instance affects only one attribute of a concept's version space, and perhaps not very much. A randomly-generated instance in observational mode is likely, on average, to make changes across several attributes to the *S* or *G* boundaries. So there is a real sense in which the best way to test a theory - holding all dimensions constant except one - is the worst way to advance concept learning in this paradigm. Is all this theorizing and experimenting worth it, in terms of training items required for learning?

We ran two control experiments to investigate this. In both experiments, Ledora was constrained to be a passive observer - it did not build or experiment with a domain model. Experiment Control-I used the iterative presentation; Control-R used the random presentation.

Figure 6.10 compares how well-learned each concept was at the end of Control-I and Control-R with the corresponding experiments in which Ledora did enter theory and experimental modes. As indicated by the concept divergence indices, the concepts in Control-I and Control-R are less well-learned at the end of the experiment than the same concepts in the corresponding experiments. Specifically, both Control-I and Control-R showed approximately a 52% decrease in concept learnedness over their counterparts.

Experiment Names	Concept Names							Divergence Index
	Religious Terror	Urban Terror	High Profile	Success Demand	Political Terror	Overt Terror	Govnt Support	
Cntl-I	(0 1 2 3 0)	(3 2 1 1 3)	(3 1 0 0)	(3 0 0 0)	(0 1 1 3)	(3 4 2 3 1)	(2 0 1 0)	44
I-C	(0 0 0 0 1)	(0 0 0 0 2)	(0 5 0 0)	(0 3 0 0)	(0 0 0 2)	(0 0 3 0 0)	(0 0 0 0)	16
I-NC	(1 1 0 0 0)	(0 0 0 0 0)	(2 5 1 1)	{(0 1 3 0) (1 0 3 0)}	(1 3 0 0)	(0 1 0 1 0)	(2 0 0 0)	27

a) Control-I versus I-C/I-NC.

Experiment Names	Concept Names							Divergence Index
	Religious Terror	Urban Terror	High Profile	Success Demand	Political Terror	Overt Terror	Govnt Support	
Cntl-R	(0 3 1 3 0)	(0 0 2 0 0)	(0 2 0 5)	(3 0 0 0)	(3 1 1 1)	(3 4 0 4 4)	(3 0 2 4)	49
R-C	(4 4 0 3 0)	(0 0 0 1 0)	(1 1 0 0)	(0 1 1 0)	(0 0 0 1)	(2 1 2 0 0)	(0 0 0 0)	22
R-C'	(0 0 1 0 0)	(1 0 0 0 0)	(1 0 1 0)	{(1 3 1 1) (3 3 0 1) }	(0 0 2 0)	(0 0 4 2 0)	(0 1 0 1)	27

R-NC	(0 1 0 0 0)	(0 0 0 0 0)	(2 1 0 0)	(1 0 0 1)	(1 0 1 4)	(0 0 0 0 0)	{
							(2 0 0 0)
							(1 0 0 2)
							(5 0 0 1)
							}
								26

R-NC'	(0 0 0 0 0)	(1 0 0 1 0)	(0 2 3 2)	(1 1 0 2)	(0 0 1 1)	(1 0 1 2 1)	(0 0 0 0)	20

b) Control-R versus R-C/R-C'/R-NC/R-NC'.

Figure 6.10 Control Experiments versus Others.

Another way to evaluate the worth of spending instances on theorizing is to subtract theory trials from the 100-trial limit. The remaining number of trials corresponds somewhat to a situation in which Ledora is simply given credible domain relations at various times to use for instance generation (in experimental mode) during learning. Ledora spent, on average, 43 of its 100 trials in theory mode. Thus, it learned more with domain relations in 57 trials than it did in 100 trials of observing instances. In general, these results show that learning proceeds faster when Ledora diverts from its primary task of learning the domain concepts to acquire predictive relations than when it focuses its attention on instances provided by the world.

A valid criticism of the above comparison is that it pits an active learner (one that can generate its own test instances) against a passive learner (one that can only observe). With just a little knowledge, an active learner can ask to see particular instances that will allow it to very quickly zero in on a concept definition. That is essentially how Ledora behaves as an experimenter. Thus, we must also evaluate Ledora's theorizing and experimenting effort with respect to just using the midpoint method. The midpoint method exploits only knowledge of the version space and the generalization hierarchies. Just as an active Ledora, armed with domain relation knowledge, is bound to outdo a passive, uninformed Ledora, this active Ledora is bound to lose when compared with the midpoint method. This is because, on average, it observes the world for 36 trials in order to propose predictive relations, and then "spends" another 43 trials in theory mode on generating instances that are definitely non-optimal for version space convergence. As pointed out previously, a theory mode instance changes only one attribute to verify a proposed predictive relation. Thus, these instances typically result in far less than 50% convergence. If, in contrast, Ledora uses only the midpoint method to generate test instances, each instance will cause a 50% convergence of a version space on some attribute for some concept. With midpoint method, every question is an optimal question in terms of advancing the concept definitions. This is why we claim that a theorizing Ledora is likely to do worse than an algorithmic (midpoint) Ledora.

All these things considered, Ledora's observe-theorize-experiment performance compared with midpoint method is not that bad. Using just the midpoint method, it took Ledora 88 trials to learn the entire set of concept definitions completely. After 100 trials as a theorizer-experimenter in the I-C run, Ledora had completely learned one concept (Government-Support) and the remaining concepts were completely learned except for one attribute each (see Figure 6.10a). So it came very close to completely learning the concepts on an extra 12 trials, and it had acquired a set of domain relations as well. There are several other advantages for having a domain model in the form of predictive relations for a learning-by-example system, which we consider in the General Discussion.

One small change to the task queue in theory mode might improve this contrast with midpoint method. Presently, Ledora executes all verification tasks before any generalization task. The rationale was that a proposed predictive relation, that was known to lead to some useful experiments, was worth investigating more than a generalization step. However, the proposed generalizations could easily be evaluated in the same fashion as the proposed predictive relations, i.e., if they were true, what impact would they have across the partially-learned concepts? Under the current scheme, some generalization that could have a huge impact on advancing several concept definitions is not investigated until all the other verifications are done. By the time it is investigated, it may be too late for it to have any positive affect on learning. By changing the theory task queue to favor investigating any predictive relation, be it for verification or for generalization, Ledora might get more mileage out of its abstracted domain relations.

Another change that might improve the performance of the observe-theorize-experiment framework involves one condition for generating experiments. Recall that one of the conditions for applying a verified predictive relation in experimental mode is that the instance generated must converge the predictable attribute by *more* than 50%. This condition may cause poor learning performance during the late stages of learning a concept description. Specifically, when there is only one state between *G* and *S* in the version space, a predictive relation will be applied to generate an experiment only if its predictable equals *S*. The chance that such a predictive relation exists is small. In such cases, Ledora usually retires to observational mode, passively observing instances from the world, despite the existence of predictive relations whose predictable is the midpoint between *G* and *S*. Hence, one improvement is to relax the above constraint so that the instance generated needs to converge the predictable attribute by *at least* 50%. This should improve Ledora's learning performance by applying more predictive relations that it knows about, hence reducing the number of instances in observational mode.

6.8 Simplifying System Parameters

As noted in Chapter 5, the Ledora system includes a set of parameters. Although this parameter set does not constitute the main design, the size of the set and the potentially complex interactions among its members can be a little disturbing. Hence, we decided to attempt to derive a simpler set of parameters. After arriving at a more simplified parameter set, we reran one of the previous experiments with this new set, and compared the results with those from the original experiment. To keep it in perspective, we admit that this is a post hoc analysis and evaluation of the parameters, and not a complete one at that. The following explains the new parameter set and the experiment performed.

Three parameters in the old set were changed: Minimum-Propagate, Minimum-Confidence and Minimum-among-Siblings. As explained before, Minimum-Propagate specifies the minimum propagation of frequency that Ledora will perform in the frequency table. The value of this parameter was modified from .01 to 0. In essence, we instructed Ledora to propagate all frequencies regardless of the amount.

Minimum-Confidence is a system parameter stating the minimum confidence that a proposed predictive relation must reach before Ledora will apply the predictive relation to generate training instances. Under the old scheme, Minimum-Confidence has a higher value than another system parameter, Minimum-Frequency, that specifies the minimum frequency of a value pair before it can emerge from the frequency table as a proposed predictive relation. In other words, after a proposed predictive relation emerges from the frequency table, it may not be available for instance generation until its confidence reaches Minimum-Confidence. To simplify this situation, we set Minimum-Confidence to the same value as Minimum-Frequency. As a result, Ledora can now investigate a proposed predictive relation as soon as it appears.

The parameters Minimum-among-Siblings and Minimum-over-Siblings together determine whether to propose a predictive relation *A:a --> B:b* by checking whether *A:a* co-occurs more frequently with *B:b* than with the siblings of *B:b*. The role of Minimum-over-Siblings is to ensure that the frequency of *{A:a B:b}* is considerably higher than the average frequency for all *{A:a B:b'}* where *B:b* and *B:b'* are siblings. On the other hand, the original intent of Minimum-among-Siblings is to complement Minimum-over-Siblings in cases where most siblings have high frequencies while a few have low frequencies. In such cases, the low frequencies drop the average below that of most siblings. Hence, the frequency for *{A:a B:b}* may be lower than most *{A:a B:b'}* even if it is above the average. Hence, we concluded at the time that the average alone is not sufficient to determine if *A:a* co-occurs more frequently with *B:b* than with the siblings of *B:b*. To rectify this situation, we introduced the parameter Minimum-among-Siblings that requires the frequency of *{A:a B:b}* to be larger than the frequencies of a specific proportion of the siblings of *B:b*, that is, of a certain percentile. However, we later discovered that Minimum-over-Siblings actually does subsume Minimum-among-Siblings. In cases with uneven frequency distribution, we can compensate for the overly-low average by increasing the minimum amount that the frequency of *{A:a B:b}* must be greater than the average. Hence, the parameter Minimum-among-Siblings is omitted in the new parameter set.

To evaluate the new system parameters, we reran experiment I-NC using only the new parameters. We call this experiment Simplified-1(S-1). Figure 6.11 compares the results of I-NC and S-1. The major difference between the two experiments is that S-1 spent considerably less time in theory mode than I-NC did. Specifically, while the number of credibility testing cases in theory mode remains equal in both experiments, the number of non-exclusive sibling generalizations has declined 91% from 11 in I-NC to 1 in S-1. This decline seems to be due to the modified parameter Minimum-Propagate. Recall that the value of this parameter was modified in order to lift the minimum-size restriction for frequency propagation. Because all frequency propagation (regardless of its size) is now permissible, higher-level value pairs in the frequency table accumulate more frequency through propagation from below. These inflated frequency counts make easier the detection of the corresponding potential predictive relations from the frequency table. This indirectly decreases the number of generalization attempts in theory mode because predictive relations that would otherwise be proposed in generalization were instead abstracted from the frequency table due to unrestricted propagation.

Because S-1 has only one non-exclusive sibling generalization, the fact that 100% of such generalization is "good" does not yield much valuable information. Aside from the decrease of generalization activities in theory mode, the two experiments produced similar results. The number and the nature of the predictive relations proposed are approximately the same in both S-1 and I-NC. Moreover, the domain concepts are about equally learned at the end of the experiments.

Experiment Names	% Distribution of Instances			% Distribution of Work in Theory Mode		Success Rate in Theory Testing (%)	% of "good" Sibling Generalizations		Success Rate in Experim Mode(%)
	Theory	Experim	Observ	Generalizing	Testing		Exclusive	Non-excl.	
I-NC	52	19	29	63	37	21	20	27	90
S-1	22	21	57	14	86	21	22	100	82

a) Mode Results.

Experiment Names	Total # of Relations	Target Relations			Transitive Relations			Spurious Relations		
		% of Total	Ave Confid	Median	% of Total	Ave Confid	Median	% of Total	AveConfid	Median
I-NC	311	16	.57	.57	-	-	-	84	.60	.62
S-1	315	17	.63	.64	-	-	-	83	.64	.67

b) Predictive Relation Results.

Experiment Names	Concept Names							Divergence Index
	Religious Terror	Urban Terror	High Profile	Success Demand	Political Terror	Overt Terror	Govnt Support	
I-NC	(1 1 0 0 0)	(0 0 0 0 0)	(2 5 1 1)	{(0 1 3 0) (1 0 3 0) }	(1 3 0 0)	(0 1 0 1 0)	(2 0 0 0)	27

S-1	(0 1 0 0 0)	(0 0 0 0 0)	(2 0 1 0)	(0 1 0 1)	(0 0 0 3)	{
						(0 0 0 1 1)
						(0 0 0 5 0)
						}
							(2 0 0 0)	18

c) Version Space Results.

Figure 6.11 Simplified versus Non-Simplified Parameters.

To conclude, the old set and the new set of parameters generated about the same results in the experiment we ran. However, far more pre-analysis is needed in the design of Ledora in order to determine crucial interactions among the system parameters and to arrive at some simpler set.

Chapter 7

General Discussion

In this thesis, we present a framework for abstracting, testing, and experimenting with predictive relations. The ultimate goal of the observe-theorize-experiment framework is to expedite learning of multiple related domain concepts. The original motivation for this research was a control issue: given several concepts to be learned, what would be an appropriate strategy for determining which concept to learn first. This assumes that the learner is able to take a proactive role in learning, e.g., it can generate instances of a particular concept it has focused on, and have those instances classified by the environment. Under this scenario, the optimal control strategy would structure learning so that concepts whose definition helps acquiring the definition of other concepts are learned first. This is certainly valuable when a concept uses another concept in its definition. For instance, a learner should learn the definition of a wheel before it learns the definition of a car.

Suppose, however, that the definition of one concept is not used in the definition of another, or that this relation among concepts is not known to the learner. A further insight suggested that, when all concepts are from the same domain, there may be some dependency relations among attribute values in this domain that we can use to accelerate concept learning. Under this light, the issue is no longer which concept to learn next, but how attention should be divided between investigating the domain - discovering and verifying predictive relations - and acquiring the domain concept definitions. Ledora demonstrates a paradigm in which predictive relations are abstracted and compete for the learner's resources. Specifically, the version space paradigm allowed us to specify a strategy for determining when exploring the domain was "worth it", namely, when it would lead to a large enough payoff in the convergence of unlearned concepts.

In the rest of this chapter, we first discuss the generality of the observe-theorize-experiment framework. Then, we explore some remaining issues specific to this research. After that, we show the relations of this research to other work, and then point out future work and extensions to Ledora. Finally, we present a summary and our conclusions of this research.

7.1 Generality of the Observe-Theorize-Experiment Framework

We propose that Ledora's framework can be used in conjunction with any concept learning method in which it is possible to quantify the degree of learnedness of concepts. By quantifying how learned a concept is, the learner can compare which of many applications of domain relations to concepts produces the best effect in terms of learning.

The ability to quantify a concept's learnedness depends on two factors. First, there must be an explicit representation of the domain structure, e.g., generalization hierarchies for attributes. This domain structure defines a fixed search space for learning the concept definitions. Second, the learner is able to inspect, at any time during learning, the representation for the partially-learned concepts. In other words, the learner has access to its current knowledge about the concept definition that has accumulated from evidence seen to date. By knowing its current hypotheses of the concept definition, with respect to the overall search space, the learner can compute how much learning still has to be done.

We chose to use the version space approach in Ledora because it satisfies the above requirement very well. In the following, we demonstrate how the observe-theorize-experiment framework can be applied to INDUCE 1.2 [Dietterich & Michalski, 1981], a non-incremental learning algorithm based on positive instances.

INDUCE's basic algorithm can be described as generate-and-test. First, given a set of positive instances of a concept, initialize a set *H* by randomly choosing a predetermined number (say *W*) of the given positive instances. These instances are essentially the "seeds" of the search. Second, generalize each description in *H* in all possible ways, thus forming the new *H*. Third, prune all but *W* of the new *H* that fail to meet certain evaluation criteria, e.g., the number of training instances covered by the description, the number of terms in the description, and any other user-specified cost measure of the terms. Fourth, test each description in *H*, and if the description covers all training instances, move it from *H* to an output concept definition set *C*. Finally, repeat steps 2, 3 and 4 until *C* reaches a predetermined size or if *H* becomes empty.

The observe-theorize-experiment framework can be applied to INDUCE 1.2 as follows. During observational mode, because the entire set of instances is available from the beginning, the learner can scan through all the instances in order to propose domain relations. In theory mode, the learner tests the validity of the proposed domain relations by generating additional test instances and asking the world for classification. In experimental mode, instead of generating more instances, the learner may use the verified domain relations in two ways. First, the domain relations can help selecting good seeds. Second, instead of generalizing all descriptions indiscriminately, and then removing all the undesirable generalizations, the learner can use the domain relations to guide its decision on what to generalize, hence making the process more computationally efficient. For example, using the predictive relation *Weapon:Gun --> Crime-Location:Latin-America*, the learner can generalize the instance *{Weapon:Gun Crime-Location:Mexico-City}* to *{Weapon:Gun Crime-Location:Latin-America}*, where Latin-America is the parent of Mexico that in turn is the parent of Mexico-City.

Note that the intermediate value *Crime-Location:Mexico* is bypassed in the generalization process. The new generalization is then inserted into H (the set of hypotheses). After that, the learner continues with steps 3 and 4 of the INDUCE 1.2 algorithm (i.e., ranking and testing the hypotheses in H, respectively).

Applying the observe-theorize-experiment framework to INDUCE 1.2 produces faster learning. In the above example, using predictive relations requires only one generalization step (from Mexico-City to Latin-America) to learn the value of the attribute Crime-Location, as compared to two steps without using predictive relations (from Mexico-City to Mexico, and then from Mexico to Latin-America). More research needs to be done before more details can be provided about how our framework can apply to INDUCE 1.2 (or any other concept learning algorithms).

7.2 Level of Instance Description

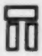
As discussed in Chapter 4, we restricted both instance descriptions generated by Ledora and by the world simulator to contain only terminal attribute values, i.e., leaf nodes in the generalization hierarchies. For instance, an instance description may include *Occupation-of-Victim:Senator* (a leaf node) but not *Occupation-of-Victim:Politician* (a non-leaf node). The main reason for imposing this restriction is more pragmatic than theoretical. Trials runs with Ledora that allowed instance descriptions to contain non-terminal values as well as terminal values showed that the version spaces converged too rapidly for the abstracted predictive relations to have an effect on learning. The generalization hierarchies given to Ledora are quite shallow (the average depth of the hierarchies is 4.6). If non-terminal values are permitted in instances, the version spaces for specific attributes may converge in two or three instances. Therefore, by the time predictive relations are abstracted and verified, the concepts' version spaces may be already or almost converged. This renders the predictive relations useless in these situations. Hence, we concluded that Ledora's paradigm is best suited for situations in which only terminal values appear in instance descriptions.

This is not an artificial constraint. Rather, it indicates the kind of learning situation in which there will be the biggest payoff for this paradigm. In some domains, the input data can only describe specific entities, rather than abstract classes. For instance, the input to the BACON and GLAUBER systems consists of specific chemical compounds, and concrete measurements taken during chemical experiments. In most cases, we must name specific substances used in each step of an experiment, e.g., add HCl to NaOH (rather than "add an acid to a base"). After all, only concrete substances can be manipulated in an experiment; abstract classes belong to theoretical discussions.

7.3 Evidence for Predictive Relations

Ledora's frequency table maintains evidence for potential predictive relations, in the form of co-occurrence information taken from the observed instances. An important issue is what constitutes evidence for predictive relations. Specifically, Ledora deals with three categories of instances, namely, positive instances of concepts, negative instances of concepts, and nonexistent instances. Which of these three categories of instances contribute towards evidence about predictive relations between attributes?

Nonexistence is a semantic notion that we introduced in this work to enable the detection and testing of predictive relations. However, to the version space method, there are only two categories of instances: positive and negative instances of concepts. Algorithmically, Ledora views nonexistent instances as negative instances of *all* domain concepts. Thus, a negative instance of a concept may refer to a counter-example of that concept or something that does not exist in the world at all. Whether nonexistent instances can be "provided" as actual negative instances of the domain concepts depends on what is possible in the world. The following discussion elaborates on this point.

In the physical world, it is often impossible to provide actual nonexistent objects as negative instances of any domain concept. One could argue that some negative arch examples for Winston's [1975] learning program are nonexistent. For instance,  (a structure where the lintel floats above the supporting pillars, rather than touches them) is a negative arch example, and cannot exist in reality because it violates the law of gravity. We can *draw* such a structure on a piece of paper, but, in the physical world, we cannot create and present this kind of structure.

Once we leave physical domains and ones which the learner cannot actually fabricate something as a test instance, it is possible to imagine nonexistent instances as follows. A nonexistent instance is something that could be conceived and described but has never been experienced. Thus, in the terrorist domain, one could think of and describe a nonexistent terrorist event that includes Supplier-of-Arms=U.S. and Recipient-of-Arms=U.S.S.R., although this event has never been (and will never be) seen. Nevertheless, this type of nonexistent event can be described and presented to the world for classification. This is essentially asking the world whether or not that event is possible in the domain. Receiving the answer that this event is nonexistent is like hearing "U.S. selling arms to U.S.S.R. just never happens because countries do not supply arms to other countries that are perceived as enemies."

The world specifies, in the classification of an instance, whether the instance is negative or nonexistent. Hence, Ledora could have updated the frequency table with negative instances (as well as positive instances) but yet omitted nonexistent instances. The main issue, however, is whether constraints between attribute values in a domain are reflected in negative instances. We argue that domain constraints are indeed reflected in negative instances. This is because a negative instance of a concept, if it actually exists in the domain, is also a positive instance of another concept, and positive instances reflect the domain constraints. For example, an instance of a chair is a negative instance of a table in the furniture domain. Nevertheless, it is clear that the chair instance adheres to the constraints in the furniture domain, e.g., unless something is supported, it will fall due to gravity.

Despite the above argument that negative instances do support domain relations, we decided that the frequency table would only be updated with positive instances but not negative instances. This decision is based on the following ad hoc reason. Allowing negative instances effectively doubles the amount of co-occurrence information in the frequency table. Consequently, more predictive relations will be proposed earlier in the learning experience from the frequency table. Many of the additional predictive relations may be spurious relations. Recall that a proposed predictive relation must be tested before it is applied. Hence, the earlier detection of more spurious predictive relations will lead to a waste of trials in testing them that could otherwise be used more effectively in learning the concept definitions.

7.4 Truthfulness of Predictive Relations

Predictive relations hold true 100% of the time in the world. In other words, all concept descriptions and all instance descriptions must adhere to the restrictions set by the predictive relations. The design decisions that are based on this fact are the following. In theory and experimental modes, Ledora generates its own test instances for the purpose of testing and applying the proposed predictive relations, respectively. These test instances are passed to the world for classification. Because predictive relations are true 100% of the time, a proposed predictive relation can be rejected and discarded immediately if the classification of merely one test instance violates the expectation of the relation.

It is important for the way we have instantiated our paradigm that predictive relations hold 100% of the time. If a predictive relation does not hold true 100% of the time, then a counter-example does not disprove it. Our conceptualization of theory mode is based on the idea that a proposed predictive relation can be confirmed or denied with a well-chosen test example. When this is not so, proposed predictive relations - or any aspect of a domain model - can only be used with some degree of statistical confidence. This confidence rating in turn is based on co-occurrence patterns in past concept instances that may change in future instances. In particular, if the order in which concept instances were encountered leads to a biased (invalid) view of the domain, then the proposed predictive relations are most likely spurious, and may lead to wrong expectation of the instances generated using these relations in experimental mode. This claim is supported by the following experimental results. In experiments with chained target predictive relations, the number of spurious predictive relations proposed by Ledora is 326 if concepts were presented randomly, and is 268 if concepts were presented iteratively. Similarly, in experiments with non-chained target predictive relations, the number of spurious predictive relations is 329 if concepts were presented randomly, and is 260 if concepts were presented iteratively.

7.5 Usefulness of Predictive Relations

Although our method ensures that the test instances generated by the verified predictive relations are more informative than those generated by the midpoint method, abstracting these predictive relations in the first place requires a substantial number of trials. If these trials were spent instead on learning the domain concepts using the midpoint method, the resultant degree of concept convergence is better than when concept learning is intermixed with exploring the domain. Experimental results showed that using just the midpoint method required less than the 100-trial limit to learn all the concept definitions completely, while abstracting and using predictive relations left some concepts only partially learned by the 100-trial limit.[1] Why should any learning system bother with predictive relations?

We can identify at least two performance situations in which knowledge of predictive domain relations can be very useful. The first situation involves the performance task of classification. Instances submitted to the performance program for classification may be incompletely-specified, i.e., some attributes may be missing in the instances. Missing information can be a problem for instance classification. For example, suppose a performance program must decide whether a given instance containing *Weight:10g* and *Color:Red* is an instance of a concept defined as *{Weight:Light-Weight Color:Primary-Color Temperature:BelowZero}*. The attribute Temperature may be missing in the instance due to a number of reasons, e.g., some variables were simply not observed or reported. Thus, the classification program cannot say with 100% certainty whether the instance belongs to the concept or not. However, predictive relations can help in this case by filling in values of missing attributes that are required to classify the instances. The predictive relations, in some sense, provide an expectation of what the values of these attributes should be. Specifically, if *Weight:10g --> Temperature:BelowZero* is a known predictive relation, and *Weight:10g*, the predictor, matches the corresponding attribute value pair in the instance description, then we can augment the instance description by attaching *Temperature:BelowZero*, the predictable. The instance now contains all three attributes that characterize the concept, and therefore can be classified.

---

[1] Using predictive relations need not preclude the use of the midpoint method. A concept learning system may use the midpoint method when none of its predictive relations can do better.

Besides filling in partially-specified instances, predictive relations can be used to detect errors in instance descriptions. Given an instance, Ledora can verify whether or not the instance description is consistent with the domain model discovered so far. For instance, given a collection of medical symptoms, e.g., pulse rate and temperature, Ledora can judge whether the measurements make sense by checking their consistency with medical knowledge, e.g., high body temperature predicts high pulse rate. If the body temperature is high but the pulse rate is low, Ledora should signal that something unusual happened. Several explanations may be investigated: previous medical knowledge needs revision, or the measuring device is dubious.

7.6 Importance of Testing Predictive Relations

Predictive relations proposed by Ledora must be verified (in theory mode) before they can be applied in experimental mode. One might ask why Ledora does not bypass theory mode and enters experimental mode directly. After all, the frequency table already provides some evidence of their validity. To answer that question, we recall that the detection of predictive relations is based on co-occurrence frequencies in observed training data. The observed training data may be biased due to many reasons, e.g., the particular order of presenting concepts may propose spurious predictive relations. Therefore, Ledora is wise to generate its own data to verify the proposed predictive relations before applying them.

One may argue that, given enough training instances, Ledora's confidence in spurious predictive relations will drop below the minimum for applying them in experimental mode. Hence, even without explicit verification, spurious predictive relations will not be available to experimental mode in the long run. The question we then ask is whether it is *necessary* to explicitly verify the proposed predictive relations. We argue that, *before* their confidence drops below the minimum for application in experimental mode, the presence of a large number of spurious relations will degrade learning performance. Specifically, these spurious predictive relations compete with valid predictive relations for applying in experimental mode, thus preventing some valid relations from being used. To avoid the situation of waiting for statistical evidence to accumulate, we decided that Ledora will explicitly verify proposed predictive relations.

7.7 Relations to Other Work

The notion of a predictive relation in our framework is similar to some of Lebowitz's [1983] notions. Lebowitz's main concern was twofold. Given a set of unclassified instances, the learner must categorize and generalize these instances into different concepts. Second, the learner must be able to access efficiently the generalized concepts in order to classify new instances. To achieve the above, Lebowitz used a data structure called the Generalization-Based Memory (GBM). A GBM is a network of nodes and arcs. Nodes represent similarities among concepts, while arcs are retrieval paths that organize concepts according to their important differences (i.e., distinguishing features). For instance, given the parent node Bomb-Attack, the arcs "Place=Northern-Ireland" and "Place=Mexico" are paths to the child concept nodes Irish-Bomb-Attack and Mexican-Bomb-Attack, respectively, that differ from each other in the value of the attribute Place.

Given a new instance, the learner searches the GBM via arcs for a generalization that matches the values in the instance. A generalization can help understand a new instance by supplying missing information. For instance, the story of a terrorist attack in Northern Ireland may not identify explicitly the victims. This missing information can be inferred from a generalization, e.g, Irish-Bomb-Attack. However, in evaluating whether a generalization matches an instance, not all the attribute values in the generalization are treated equally. Lebowitz classified attribute values in a generalization into two types: predictors and predictables. The predictors are those that indicate the relevance of a generalization, while the predictables are those that do not. In other words, seeing the predictors of a generalization in an instance tells us that the generalization matches the instance, but seeing the predictables in an instance does not tell us any information. Hence, only the predictors are allowed as values of arcs in GBM while the predictables are just stored in nodes.

Ledora's predictors and predictables are defined in a similar way as those in GBM. In GBM, the predictors of a generalization are those that are unique, or nearly unique, in the context of other generalizations, while the predictables may appear in many other generalizations. Because the predictors are unique for each generalization, given that an instance contains the predictor, the learner can safely say that the instance belongs to the corresponding generalization, and predicts that the predictables are also true in the instance. On the contrary, given that an instance contains the predictable, the predictor may not be true in the instance. To compare Lebowitz's notion of predictability with ours, suppose $A{:}a \dashrightarrow B{:}b$ is a predictive relation in Ledora. The predictor $A{:}a$ is, in some sense, unique, in that $A{:}a$ cannot co-occur with $B{:}b'$ where $B{:}b'$ is any sibling of $B{:}b$. The predictable $B{:}b$ is not unique in that it can co-occur with any value of attribute A.

Predictive relations proposed by Ledora can be viewed as heuristics. This is because their detection is based on the evidence that the relations are true of the entire domain, and yet there is no guarantee that new domain concepts encountered later will not contradict these relations. Lenat [1982] discussed a tradeoff between generality and power of heuristics. Generality refers to the number of tasks (or situations) in which it is appropriate to apply the heuristic. Power refers to the degree of utility of applying the heuristic. The tradeoff observed by Lenat is that a powerful heuristic is often only applicable to very few tasks. In addition, applying a powerful heuristic to tasks other than the appropriate ones can have zero or even negative utility. Hence, before we apply a powerful heuristic to a task, we must make sure that the heuristic is truly appropriate to the task.

In Ledora, the important issue is not whether a proposed predictive relation is appropriate (or applicable) to a concept. This is ensured with certain pre-conditions that must be satisfied before a proposed predictive relation is considered applicable. Rather, the issue is whether the predictive relation is valid. The cost of applying an invalid predictive relation to generate a concept instance can be measured by the "missed" degree of convergence if the midpoint method were used instead. Recall that the midpoint method guarantees that each instance generated will reduce the version space by half. The amount of potential convergence lost depends on the position of the predictable in the concept's version space. If the proposed predictive relation is invalid, a predictable nearer the boundary will cause less convergence than a predictable that is nearer the midpoint. Therefore, to maximize the utility of a proposed predictive relation, Ledora must occasionally divert from the primary task of learning concept definitions in order to verify the validity of the predictive relation.

Ledora incorporates two different paradigms of learning: learning by example and learning by observation and discovery. Its primary task of learning concept definitions is essentially one of learning by example; its secondary task of acquiring domain relations is one of learning by observation and discovery. Like GLAUBER and STAHL [Langley, et al., 1986], the discovery aspects of Ledora resemble a data-driven system. In Ledora, training instances alone determine the alternative domain relations that Ledora will consider. Specifically, with each training instance, Ledora modifies appropriate co-occurrence information stored in the frequency table. The frequency table mechanism, which is essentially an inductive operator, abstracts regularities in the form of domain relations from these co-occurrence information.

7.8 Future Work and Extensions

There are three major areas for future research with Ledora. First, there must be more experimentation with Ledora. Second, Ledora can be enhanced by the capability to modify its own generalization hierarchies. Third, minor changes to Ledora's design (as suggested in Chapter 6) should be implemented and evaluated. The remainder of this chapter discusses each of the three areas in more details.

7.8.1 Further Experimentation

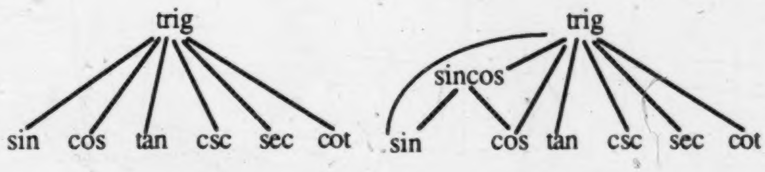
The most important direction for future research is more experimentation with Ledora. More experiments are needed to: (a) support current results and draw more conclusive results regarding random versus iterative concept presentation and chaining versus no-chaining target predictive relations, (b) substantiate claims about the usefulness of predictive relations to the performance task of classifying instances in a world with incomplete or noisy data, and (c) verify the generality of the observe-theorize-experiment framework by implementing it using INDUCE 1.2 as the underlying concept learning method.

7.8.2 Modifying Generalization Hierarchies

Generalization hierarchies constitute a bias in the search for domain concept definitions. They comprise a language in which domain concept definitions are expressed. The hierarchies used in most concept learning systems are static, i.e., they cannot be changed during the course of learning. An exception is STABB [Utgoff, 1986], a concept learning system capable of shifting its bias in terms of deriving a new concept description language while learning concept definitions. STABB uses the version space method for concept learning. It modifies the concept description language when a version space becomes empty, meaning that no hypothesis is consistent with all the training data seen so far. One method that STABB uses to search for a better bias involves the least-specific disjunction of existing descriptions. For example, suppose that

$$\int x \sin(x)\, dx, \text{ and}$$
$$\int x \cos(x)\, dx$$

are two positive instances and that $\int x \tan(x)\, dx$ is a negative instance for applying the Integration-by-Parts operator. Given the simple language depicted in Figure 7.1a, there is no description in that language that is consistent with all three instances. STABB modifies the language by introducing into the language the new term sincos(x) that effectively means sin(x) ∨ cos(x). Figure 7.1b shows the language after the shift.



a) Before b) After

Figure 7.1 Bias Before and After Shift.

At present, Ledora is not capable of dynamically proposing and making changes to its generalization hierarchies. One future research direction may be to provide Ledora with the capability of modifying its own knowledge base. This is a reasonable approach because the hierarchies pre-defined to Ledora may be incomplete or inaccurate, and as Ledora gradually develops a model for the domain, it may come to realize the deficiencies in its concept description language.

We now illustrate one situation where Ledora might consider changing its generalization hierarchies. Suppose that b_1, b_2, b_3 are all the immediate specializations of b of the attribute B, and that Ledora currently knows about the following predictive relations: $A{:}a \dashrightarrow B{:}b_1$, and $A{:}a \dashrightarrow B{:}b_2$. Knowing the two predictive relations prompts Ledora to propose a generalization of these relations to $A{:}a$ --> $B{:}b$. Test instances generated to see if $A{:}a$ indeed predicts $B{:}b_3$ as well as $B{:}b_1$ and $B{:}b_2$ (hence testing $A{:}a$ --> $B{:}b$) may fail. This serves as an indirect indication that $B{:}b_1$ and $B{:}b_2$ have something in common that accounts for why they are predicted by $A{:}a$, given b_3 is not. In other words, their presence in these predictive relations may signal the need to create some intervening subclass of b in the knowledge base, which has b_1 and b_2 as members, but not b_3. This corresponds to STABB's technique of least-specific disjunction of existing descriptors.

7.8.3 Design Changes

The third step to be taken with Ledora is making some minor design changes. As mentioned in section 6.7, Ledora's learning performance can be improved by two simple design changes. The first change involves modifying the priority scheme for selecting tasks for execution in theory mode. Specifically, instead of assigning a higher priority to verification tasks over generalization tasks, Ledora should assign the highest priority to the task (be it verification or generalization) that has the most impact on learning. The second change is to allow instances generated using verified predictive relations in experimental mode to converge the predictable attribute by *at least* 50%.

7.9 Summary and Conclusions

Ledora demonstrates that, while learning the definitions of multiple domain concepts, a learning system can simultaneously abstract important relations among values of the domain attributes. Moreover, these domain relations can in turn be used to help learning the concept definitions. To co-ordinate learning domain concepts and exploring the domain, Ledora uses a control strategy that we term the observe-theorize-experiment framework. This framework is best suited for learning situations in which (a) the learner can present examples to the world for classification; (b) domain relations hold true 100% of the time; (c) instances are described with specific attribute values and not classes; and (d) there are uses for the resulting domain relations outside the learning task (e.g., to act as expectations to fill in missing instance information or to indicate potential errors in instance descriptions). Preliminary experimental results showed that the observe-theorize-experiment framework compares favorably in terms of learning performance with alternative methods that do not use domain relations. To conclude, we believe that the observe-theorize-experiment framework is a good candidate for a general paradigm of multiple concept learning.

Bibliography

Amarel, S.(1986). Program Synthesis as a Theory Formation Task: Problem Representations and Solution Methods. In R.S. Michalski, J.G. Carbonell and T.M. Mitchell(Eds), Machine Learning: An Artificial intelligence Approach, Vol 2. Morgan Kaufmann: Los Altos.

Brown, J.S.(1973). Steps Toward Automatic Theory Formation. Proc. 3rd IJCAI, Stanford, CA.

Buchanan, B.G. & Feigenbaum, E.A.(1978). DENDRAL and Meta-DENDRAL: Their Applications Dimension, Artificial Intelligence, 11, 5-24.

Buchanan, B.G. & Mitchell, T.M.(1978). Model-Directed Learning of Production Rules. In D.A. Waterman and F. Hayes-Roth(Eds), Pattern-Directed Inference Systems. Academic Press: New York.

Buchanan, B.G., Smith, D.H., White, W.C., Gritter, R.J. , Feigenbaum, E.A., Lederberg, J. & Djerassi, C.(1976). Applications of Artificial Intelligence for Chemical Inference. 22. Automatic Rule Formation in Mass Spectrometry by Means of the Meta-DENDRAL Program, J. American Chemical Society, 98:20.

Carbonell, J.G. & Gil, Y.(1987). Learning by Experimentation. Proc 4th Machine Learning Workshop, Irvine, CA.

Carbonell, J.G., Michalski, R.S. & Mitchell, T.M.(1983). An Overview of Machine Learning. In R.S. Michalski, J.G. Carbonell and T.M. Mitchell(Eds), Machine Learning: An Artificial intelligence Approach, Vol 1. Tioga: Palo Alto.

Cohen, P.R., & Feigenbaum, E.A.(1982). The Handbook of Artificial Intelligence, Volume 3. William Kaufmann: Los Altos.

Dietterich, T.G. & Michalski, R.S.(1981). Inductive Learning of Structural Descriptions: Evaluation Criteria and Comparative Review of Selected Methods, Artificial Intelligence, 16.

Dietterich, T.G. & Michalski, R.S.(1986). Learning to Predict Sequences. In R.S. Michalski, J.G. Carbonell and T.M. Mitchell(Eds), Machine Learning: An Artificial intelligence Approach, Vol 2. Morgan Kaufmann: Los Altos.

Emde, W., Habel, C.U. & Rollinger C.(1983). The Discovery of the Equator or Concept Driven Learning. Proc. IJCAI.

Falkenhainer, B.(1985). Proportionality Graphs, Units Analysis and Domain Constraints: Improving the Power and Efficiency of the Scientific Discovery Process. Proc. IJCAI.

Feigenbaum, E.A. & Feldman, J.(1963). Computers and Thought. McGraw-Hill Inc., New York.

Forbus, K.D. & Gentner, D.(1986). Learning Physical Domains: Toward a Theoretical Framework. In R.S. Michalski, J.G. Carbonell and T.M. Mitchell(Eds), Machine Learning: An Artificial intelligence Approach, Vol 2. Morgan Kaufmann: Los Altos.

Forgy, C.L.(1979). OPS4 User's Manual. Technical Report CMU-CS-79-132. Carnegie-Mellon University.

Jones, R.(1986). Generating Predictions to Aid the Scientific Discovery Process. Proc. AAAI.

Kodratoff, Y. & Tecuci, G.(1987). Disciple-1: Interactive Apprentice System in Weak Theory Fields. Proc. IJCAI.

Langley, P.(1981). Data-driven Discovery of Physical Laws. Cognitive Science 5, 31-54.

Langley, P., Bradshaw, G.L. & Simon H.A.(1981). BACON.5: The Discovery of Conservation Laws. Proc. IJCAI.

Langley, P., Bradshaw, G.L. & Simon H.A.(1983). Rediscovering Chemistry with the BACON system. In R.S. Michalski, J.G. Carbonell and T.M. Mitchell(Eds), Machine Learning: An Artificial intelligence Approach, Vol 1. Tioga: Palo Alto.

Langley, P., Zytkow, J.M., Simon H.A. & Bradshaw, G.L.(1986). The Search for Regularity: Four Aspects of Scientific Discovery. In R.S. Michalski, J.G. Carbonell and T.M. Mitchell(Eds), Machine Learning: An Artificial intelligence Approach, Vol 2. Morgan Kaufmann: Los Altos.

Lebowitz, M.(1983). Generalization from Natural Language Text, Cognitive Science, 7, 1-40.

Lebowitz, M.(1986). Concept Learning in a Rich Input Domain: Generalization-Based Memory. In R.S. Michalski, J.G. Carbonell and T.M. Mitchell(Eds), Machine Learning: An Artificial intelligence Approach, Vol 2. Morgan Kaufmann: Los Altos.

Lenat, D.B.(1979). On Automated Scientific Theory Formation: A Case Study using the AM Program. In J.E. Hayes, D.Michie and L.I. Mikulich(Eds), Machine Intelligence, Vol 9. Halsted: New York.

Lenat, D.B.(1982). The Nature of Heuristics, Artificial Intelligence, 19, 189-249.

Lenat, D.B.(1983a). Theory Formation By Heuristic Search: the Nature of Heuristics II: Background and Examples, Artificial Intelligence, 21, 31-59.

Lenat, D.B.(1983b). EURISKO: a Program that Learns New Heuristics and Domain Concepts: the Nature of Heuristics III: Program Design and Results, Artificial Intelligence, 21, 61-98.

Lenat, D.B.(1983c). The Role of Heuristics in Learning by Discovery: Three Case Studies. In R.S. Michalski, J.G. Carbonell and T.M. Mitchell(Eds), Machine Learning: An Artificial intelligence Approach, Vol 1. Tioga: Palo Alto.

Michalski, R.S.(1986). Understanding the Nature of Learning: Issues and Research Directions. In R.S. Michalski, J.G. Carbonell and T.M. Mitchell(Eds), Machine Learning: An Artificial intelligence Approach, Vol 2. Morgan Kaufmann: Los Altos.

Michie, D.(1987). Current Developments in Expert Systems. In J.R. Quinlan(Ed), Applications of Expert Systems. Addison-Wesley.

Minton, S.N., Carbonell, J.G., Etzioni, O., Knoblock, C.A. & Kuokka, D.R.(1987). Acquiring Effective Search Control Rules: Explanation-Based Learning in the PRODIGY System. Proc. 4th Machine Learning Workshop, Irvine, CA.

Mitchell, T.M.(1977). Version Spaces: A Candidate Elimination Approach to Rule Learning. Proc. IJCAI.

Mitchell, T.M.(1982). Generalization as Search, Artificial Intelligence, 18, 203-226.

Mitchell, T.M., Utgoff, P.E. & Banerji, R.B.(1983). Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics. In R.S. Michalski, J.G. Carbonell and T.M. Mitchell(Eds), Machine Learning: An Artificial intelligence Approach, Vol 1. Tioga: Palo Alto.

Mitchell, T.M., Utgoff, P.E., Nudel, B. & Banerji, R.B.(1981). Learning Problem-Solving Heuristics Through Practice. Proc. IJCAI.

Nordhausen, B. & Langley, P.(1987). Towards an Integrated Discovery System. Proc. IJCAI.

Pazzani, M., Dyer, M. & Flowers, M.(1986). The Role of Prior Causal Theories in Generalization. Proc. AAAI.

Pazzani, M., Dyer, M. & Flowers, M.(1987). Using Prior Learning to Facilitate the Learning of New Causal Theories. Proc. IJCAI.

Poole, D., Goebel, R., & Aleliunas, R.(1986). Theorist: a Logical Reasoning System for Defaults and Diagnosis. Tech. Report CS-86-06. University of Waterloo.

Quinlan, J.R.(1979). Discovering Rules by Induction from Large Collections of Examples. In D. Michie(Ed), Expert Systems in the Micro-Electronic Age. Edinburgh University Press: Edinburgh.

Quinlan, J.R.(1983). Learning Efficient Classification Procedures and their Application to Chess End Games. In R.S. Michalski, J.G. Carbonell and T.M. Mitchell(Eds), Machine Learning: An Artificial intelligence Approach, Vol 1. Tioga: Palo Alto.

Quinlan, J.R.(1986). The Effect of Noise on Concept Learning. In R.S. Michalski, J.G. Carbonell and T.M. Mitchell(Eds), Machine Learning: An Artificial intelligence Approach, Vol 2. Morgan Kaufmann: Los Altos.

Quinlan, J.R., Compton, P.J., Horn K.A. & Lazarus, L.(1987). Inductive Knowledge Acquisition: a Case Study. In J.R. Quinlan(Ed), Applications of Expert Systems. Addison-Wesley.

Rajamoney, S. & DeJong G.(1987). The Classification, Detection and Handling of Imperfect Theory Problems. Proc. IJCAI.

Romanycia, M.H.J. & Pelletier F.J.(1985). What is a Heuristic? Computational Intelligence 1, 47-58.

Russell, S.J.(1986). Preliminary Steps Toward the Automation of Induction. Proc. AAAI.

Subramanian, D. & Feigenbaum, J.(1986). Factorization in Experiment Generation. Proc. AAAI.

Utgoff, P.E.(1986). Shift of Bias for Inductive Concept Learning. In R.S. Michalski, J.G. Carbonell and T.M. Mitchell(Eds), Machine Learning: An Artificial intelligence Approach, Vol 2. Morgan Kaufmann: Los Altos.

Winston, P.H.(1975). Learning Structural Descriptions from Examples. In P.H. Winston(Ed), The Psychology of Computer Vision. McGraw-Hill Inc., New York.

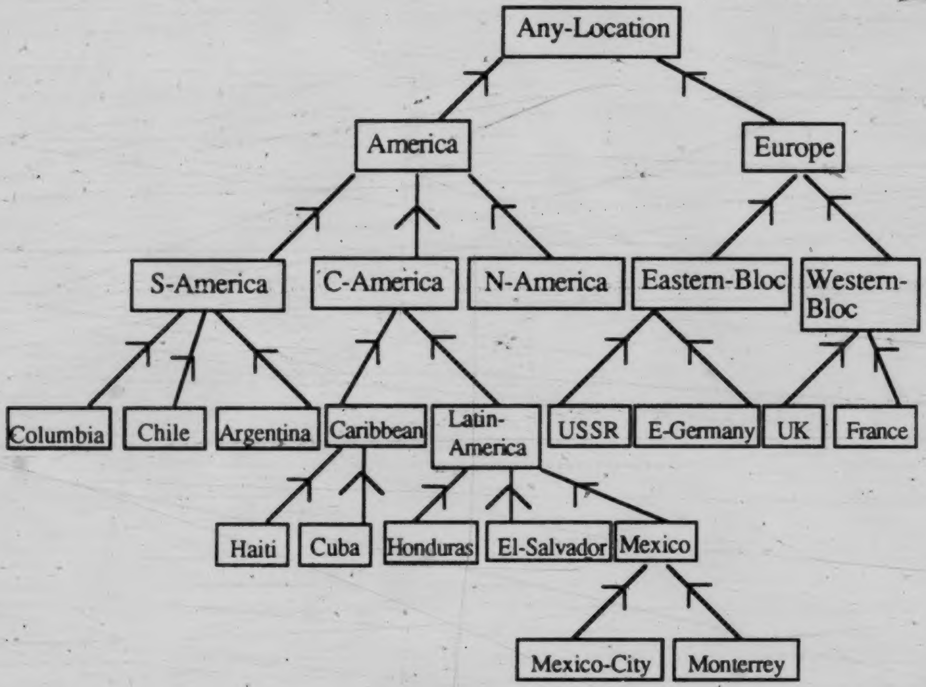
Zytkow, J.M.(1987). Combining Many Searches in the FAHRENHEIT Discovery System. Proc. 4th Machine Learning Workshop, Irvine, CA.

Zytkow, J.M.& Simon H.A.(1986). A Theory of Historical Discovery: the Construction of Componential Models. Machine Learning 1, 107-137.

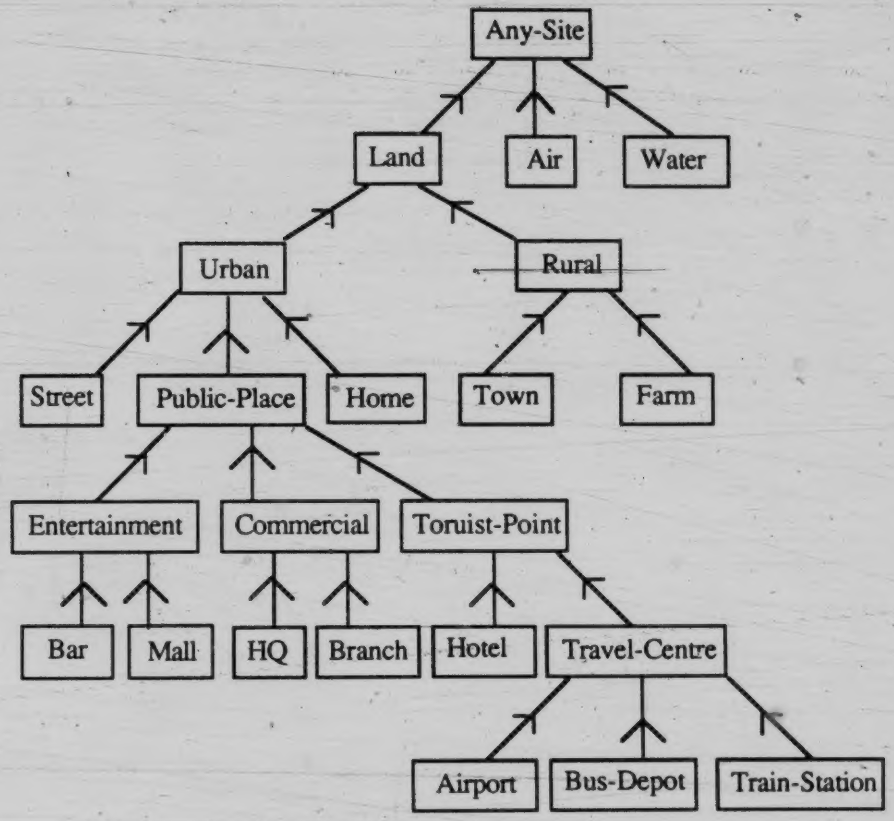
Appendix

Generalization Hierarchies

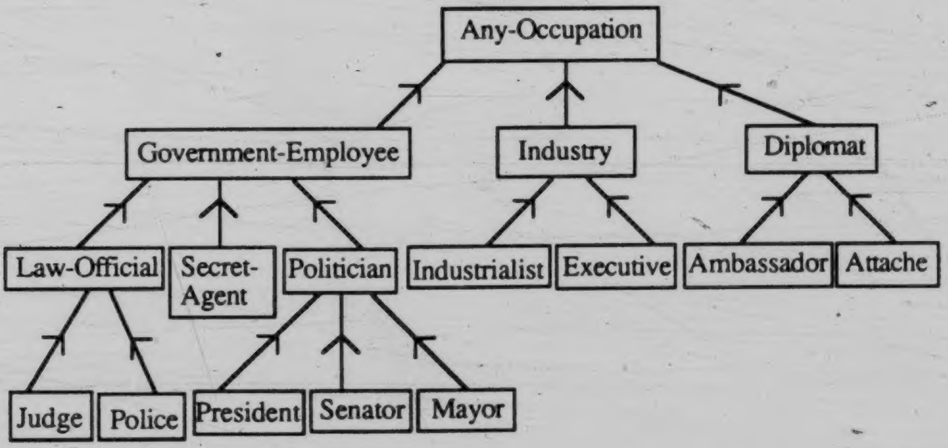
This appendix depicts the generalization hierarchies for all attributes used in our experiments in alphabetical order.


Any-Location
America
Europe
S-America
C-America
N-America
Eastern-Bloc
Western-Bloc
Columbia
Chile
Argentina
Caribbean
Latin-America
USSR
E-Germany
UK
France
Haiti
Cuba
Honduras
El-Salvador
Mexico
Mexico-City
Monterrey

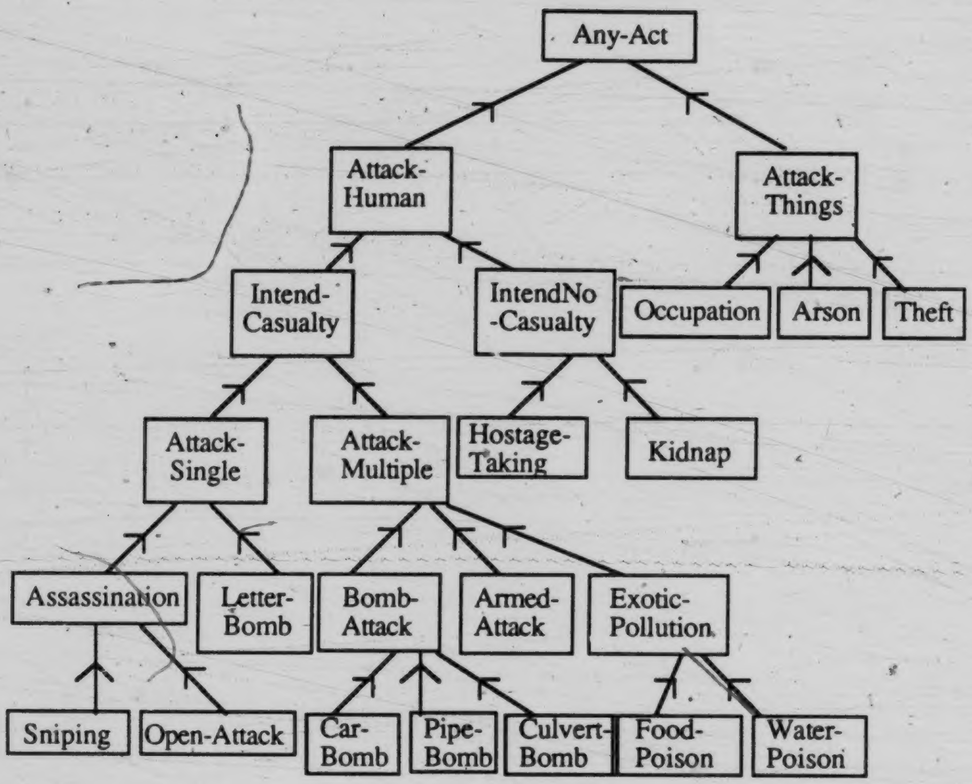
(a) Crime-Location


Any-Site
Land
Air
Water
Urban
Rural
Street
Public-Place
Home
Town
Farm
Entertainment
Commercial
Toruist-Point
Bar
Mall
HQ
Branch
Hotel
Travel-Centre
Airport
Bus-Depot
Train-Station

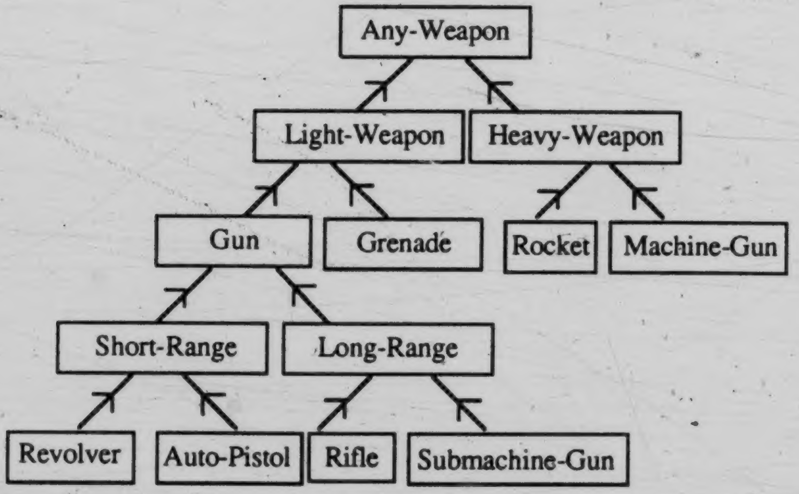
(b) Crime-Site


Any-Occupation
Government-Employee
Industry
Diplomat
Law-Official
Secret-Agent
Politician
Industrialist
Executive
Ambassador
Attache
Judge
Police
President
Senator
Mayor

(c) Occupation-of-Victim


Any-Act
Attack-Human
Attack-Things
Intend-Casualty
IntendNo-Casualty
Occupation
Arson
Theft
Attack-Single
Attack-Multiple
Hostage-Taking
Kidnap
Assassination
Letter-Bomb
Bomb-Attack
Armed-Attack
Exotic-Pollution
Sniping
Open-Attack
Car-Bomb
Pipe-Bomb
Culvert-Bomb
Food-Poison
Water-Poison

(d) Type-of-Activity


Any-Weapon
Light-Weapon
Heavy-Weapon
Gun
Grenade
Rocket
Machine-Gun
Short-Range
Long-Range
Revolver
Auto-Pistol
Rifle
Submachine-Gun

(e) Weapon